

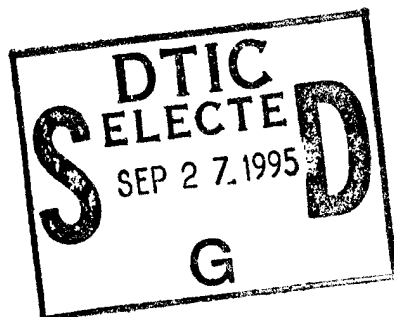
PROCEEDINGS

IFIP WG 11.3

Sixth Working Conference

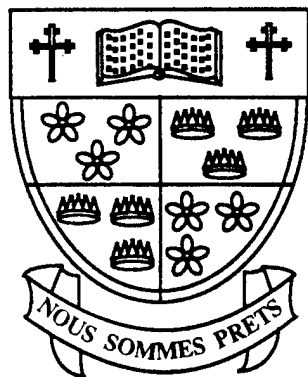
on

DATABASE SECURITY



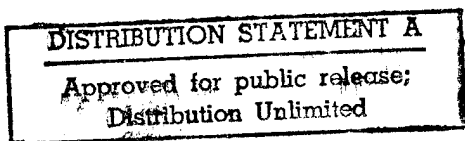
19950922 069

DTIC QUALITY INSPECTED 5



19-21 August 1992

*Simon Fraser University
Burnaby, Vancouver, British Columbia*



194 7 8 002



OFFICE OF THE UNDER SECRETARY OF DEFENSE (ACQUISITION)
DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22304-6145

July 13, 1994

IN REPLY
REFER TO

DTIC-OCC

SUBJECT: Distribution Statements on Technical Documents

TO: Office of the Chief of Naval Research
800 north Quincy Street
Arlington, VA 22217-5000
Code 22

1. Reference: DoD Directive 5230.24, Distribution Statements on Technical Documents, 18 Mar 87.

2. The Defense Technical Information Center received the enclosed report (referenced below) which is not marked in accordance with the above reference.

Final Technical Report
N00014-92-J-1952
19 - 21 August 1992

3. We request the appropriate distribution statement be assigned and the report returned to DTIC within 5 working days.

4. Approved distribution statements are listed on the reverse of this letter. If you have any questions regarding these statements, call DTIC's Cataloging Branch, (703) 274-6837.

FOR THE ADMINISTRATOR:

1 Encl

Gopalakrishnan Nair
GOPALAKRISHNAN NAIR
Chief, Cataloging Branch

FL-171
Jul 93

1995 0922 069

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES ONLY;
(Indicate Reason and Date Below). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED
TO (Indicate Controlling DoD Office Below).

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND THEIR CONTRACTORS;
(Indicate Reason and Date Below). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED
TO (Indicate Controlling DoD Office Below).

DISTRIBUTION AUTHORIZED TO DOD AND U.S. DOD CONTRACTORS ONLY; (Indicate Reason and Date Below). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION AUTHORIZED TO DOD COMPONENTS ONLY; (Indicate Reason and Date Below).
OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

FURTHER DISSEMINATION ONLY AS DIRECTED BY (Indicate Controlling DoD Office and Date Below) or HIGHER LOD AUTHORITY.

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND PRIVATE INDIVIDUALS OR ENTERPRISES ELIGIBLE TO OBTAIN EXPORT-CONTROLLED TECHNICAL DATA IN ACCORDANCE WITH DOD DIRECTIVE 5230.25, WITHHOLDING OF UNCLASSIFIED TECHNICAL DATA FROM PUBLIC DISCLOSURE, 6 Nov 1984 (Indicate date of determination). CONTROLLING DOD OFFICE IS (Indicate Controlling DoD Office).

OFFICE OF NAVAL RESEARCH
CORPORATE PROGRAMS DIVISION
ONR 353
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

(Controlling DoD Office Name)

DEBRA T. HUGHES
DEPUTY DIRECTOR
CORPORATE PROGRAMS OFFICE

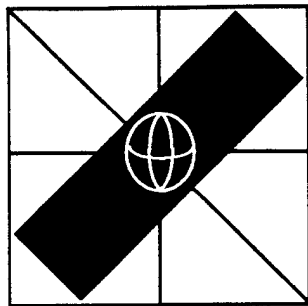
(Controlling DoD Office Address,
City, State, Zip)

15 SEP 1995

(Signature & Typed Name)

(Assigning Office)

(Date Statement Assigned)



PROCEEDINGS

IFIP WG 11.3

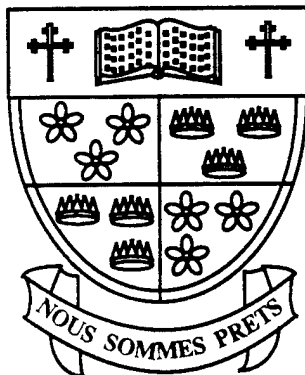
Sixth Working Conference

on

DATABASE SECURITY



Accession For	
NTIS	<input checked="" type="checkbox"/>
CRA&I	<input checked="" type="checkbox"/>
DTIC	<input type="checkbox"/>
TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



19-21 August 1992

*Simon Fraser University
Burnaby, Vancouver, British Columbia*

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited

ACKNOWLEDGMENT

I thank the following people for making the Sixth IFIP 11.3 Working Conference in Database Security a great success:

The authors of the papers submitted to the conference,

The reviewers of the papers,

Carl Landwehr, the chairman of the IFIP 11.3 Working Group, for his encouragement throughout the organization of the conference and for his support in putting the final program together,

David Bonyun, for his activities as the general chair of the conference,

Sushil Jajodia, for publishing these proceedings,

Teresa Lunt, for taking notes of the discussions,

The participants of the conference,

The sponsors of the conference, and

Matthew Morgenstern, for organizing a panel in database security at the 18th International Conference on Very Large Databases.

Bhavani Thuraisingham
Program Chair

SIXTH IFIP 11.3 WORKING CONFERENCE COMMITTEE

PROGRAM CHAIR

Bhavani Thuraisingham
The MITRE Corporation
K306
Burlington Road
Bedford, MA 01730
U.S.A.

GENERAL CHAIR

David Bonyun
Bunberry, Filbert, and Stokes
Enterprises Ltd.
7 Keppler Crescent
Nepean, Ontario K2H 5Y1
CANADA

IFIP WG11.3 CHAIR

Carl Landwehr
Naval Research Laboratory
Code 5542
4555 Overlook Ave., SW
Washington, DC 20375-5000
U.S.A.

PREFACE

These Proceedings consist of the papers presented at the Sixth IFIP Working Conference in Database Security held in Vancouver, British Columbia, from 19 to 22 August 1992. The papers cover a variety of topics in database security including multilevel semantic data models, inference problem, policies and models, multilevel database concurrency control, and multilevel relational data models.

TABLE OF CONTENTS

	PAGE
Keynote Talk	
1. <i>Protecting Sensitive Medical Information</i> Jeffrey Kaplan, M.D., Hartford, CT	1
Semantic Data Models and Multilevel Security	
2. <i>Implementing the Message Filter Object-oriented Security Model without Trusted Subjects</i> Roshan Thomas and Ravi Sandhu, George Mason University	13
3. <i>Multilevel Secure Rules: Integrating the Multilevel Secure and Active Data Models</i> Kenneth Smith and Marianne Winslett University of Illinois at Urbana Champaign	33
Multilevel Database Applications	
4. <i>The Spear Data Design Method</i> Peter Sell, Office of INFOSEC Computer Science	59
5. <i>Using Sword for the Military Aircraft Command Example Database</i> Simon Wiseman, Defense Research Agency	77
Policies and Models - I	
6. <i>Extending Access Control with Duties</i> Dirk Jonscher, Universitat Rostock	107
7. <i>Formalising and Validating Complex Security Requirements</i> Philip Morris and John McDermid, University of York	127
8. <i>Support for Security Modeling in Information Systems Design</i> Gerhard Steinke and Matthias Jarke, University of Passau	141
Inference Problem - I	
9. <i>Toward a Tool to Detect and Eliminate Inference Problems in the Design of Multilevel Databases</i> Thomas Garvey, Teresa Lunt, Xiaolei Qian, and Mark Stickel, SRI International	159
10. <i>Inference and Cover Stories</i> Leonard Binns, Office of INFOSEC Computer Science	179

TABLE OF CONTENTS (Continued)

	PAGE
Inference Problem - II	
11. <i>Aerie: An Inference Modeling and Detection Approach for Databases</i> Thomas Hinke and Harry Delugach, University of Alabama, Huntsville	187
12. <i>Inference Through Secondary Path Analysis</i> Leonard Binns, Office of INFOSEC Computer Science	203
13. <i>Disclosure Limitation using Autocorrelated Noise</i> George Duncan and Sumitra Mukherjee, Carnegie Mellon University	213
Secure Distributed Database Management Systems Panel	
Concurrency Control - I	
14. <i>Recovery Management for Multilevel Secure Database Systems</i> Iwen Kang and Thomas Keefe, The Pennsylvania State University	227
15. <i>Maintaining Multilevel Transaction Atomicity in MLS Database Systems with Kernelized Architecture</i> Oliver Costich and Sushil Jajodia, George Mason University	253
Concurrency Control - II	
16. <i>Orange Locking: Channel-Free Database Concurrency Control Via Locking</i> John McDermott, Naval Research Laboratory Sushil Jajodia, George Mason University	271
17. <i>A Practical Transaction Model and Untrusted Transaction Manager for a Multilevel Secure Database System</i> Myong Kang and Judith Froscher, Naval Research Laboratory, Oliver Costich, George Mason University	289
Multilevel Data Models	
18. <i>Tuple-level vs Element-level Classification</i> Xiaolei Qian and Teresa Lunt, SRI International	311
19. <i>Inference Secure Multilevel Databases</i> T.Y. Lin, California State University, San Jose	327

TABLE OF CONTENTS (Concluded)

Policies and Models - II

- | | | |
|-----|---|-----|
| 20. | <i>A Specification Methodology for User-Role Based Security in an Object-Oriented Design Model</i> | 351 |
| | T.C. Ting, S. Demurjian, and M.Y. Hu,
University of Connecticut , Storrs | |
| 21. | <i>Integrity and the Audit of Trusted Database Management Systems</i> | 379 |
| | Jarrellann Filsinger, The MITRE Corporation | |
| 22. | <i>On the Axiomatization of Security Policy: Some Tentative Observations about Logic Representation</i> | 401 |
| | Bret Michael, Edgar Sibley, Richard Baum, and Fu Li,
George Mason University | |

PROTECTING SENSITIVE MEDICAL INFORMATION

Jeffrey Gene Kaplan, M.D., M.P.S.

148 Sunny Reach Dr.

Hartford, CT 06117

ABSTRACT

Managed Care combines medical utilization and its cost with control of legal risk. Review of these three factors leads to salient information as well as understanding of the health care delivery processes. A point-of-contact system must operate as a relational database for data access, management, analysis and reporting.

Protection of sensitive information in the medical record is a patient's right. It is also the responsibility of the care-giver. Such medical information must not be revealed to anyone who is not entitled nor authorized. The use of encryption and masks within the structure of the database and user-defined access will provide the necessary security.

The new approach to quality health service and cost effectiveness is to recognize the intelligence of patients and their desire to be active participants in their care. When presented with a choice of procedures and the probable outcome of each, patients are empowered to make informed decisions.

DEVELOPING INFORMATION FOR MANAGING COST AND QUALITY OF HEALTH CARE

In developing a health information technology, there are four cardinal questions: 1) how do you get to data, 2) how do you translated these data into information, 3) how do you deliver it effectively without betraying a person's confidentiality, and finally, 4) how do you know you are capturing the right data in the first place?

A basic lesson learned from analyzing computer applications for health information is the importance of confidentiality. It is most efficient, therefore, to design both the information and security systems together¹.

THE SENSITIVITY OF MEDICAL INFORMATION

The exchange of information is an ethical issue. The focus must be on the interests and rights of those connected to the information and on the monitors of the organizations where data are stored or processed. Rather than simply protecting sensitive information, control of access to these data should be based upon one's legitimate role. Unfortunately, security of medical data is not common-place in the research and development community. This may be due to the antiquated view that the essential business of health care is the administration of benefits, insurance and coverage. This leads to assigning blame for the health care crisis to the physicians who order the covered tests and treatments.

I use the analogy of car manufacture and Japanese quality control. When the painting of the car is imperfect, the Japanese, instead of repainting it, will change the painting procedure. This is how one improves quality.

This idea applied to medical care is too simplistic, however. What medical management needs and deserves is better information about what works best in health care. From data to information, we must strive for superior quality of medical care by concentrating on both process and outcome. While we need to weed out those individuals who are of poor quality, variance analysis that only focuses on "who," rather than "what" does little for the overall improvement in the distribution¹ of the quality of medical care.

With little information about clinical effectiveness, efficiency and value,² health care managers, patients and employers have data, but little information. Therefore, the important question is what data are required to manage care effectively. We need specifics about the health care delivery system (structure), its clinical assessments and medical treatments (process), plus the results of all medical activity (outcome). Unfortunately, there is a dearth of vital clinical information available to the health analyst, and, in turn, our medical leadership. Nevertheless, the prospect of reconstructing a medical database from the claims trail, especially given recent developments in electronic data interchange³ (EDI) is encouraging. At the point-of-contact with the patient, medical informatics technology will provide immediate knowledge of who the patients are, their medical history and

¹ The statistics of outlier analysis is only lowered for quality for 1-5% of its members. If one applies the bell-shaped curve to quality, there is little shift towards improvement.

² I define "value," for these purposes, as a relationship of quality, efficacy and cost-effectiveness

³ such as electronic billing

financing mechanism (i.e., insurance eligibility⁴), the tests that have been performed and the physician(s) involved.

A NEW COMPREHENSION: A NEW DATABASE AND KNOWLEDGE-BASED SYSTEM

The health insurance and managed care industry have relied on cost data to manage care. However, these data are often unreliable and incomplete. For one thing, there is insufficient clinical context for us to account for the care. The health care crisis and our inability to rectify the situation from within suggest the need for a new, more progressive approach. A shift in paradigms² is required. I would call the new approach -- informed decision making (IDM³). This means giving people information about their condition, choice of procedures, and expected results of each. Patients can then make informed decisions about what will be done to their bodies. This is far better than merely enforcing a contract, fixing a fee, and regulating the volume of services without regard to the nature of the illness.

In the pursuit of value and IDM, the managed care industry is becoming positioned⁵ to collect, manage, analyze and present information about the quality of health care and its cost. The tools for managing clinical process must involve a clinical data base⁶. To this should be added clinical profiling, statistical analyses and quality improvement⁴ methods. These are the tools for managed care. We use them as we work with providers fighting the unexplainable variation of medical practice and the spiraling cost of health care⁵.

THE MECHANICS OF INFORMED DECISION MAKING

The mechanics of informed decision making must be explained. Five information-generating, data-transformation tools are essential:

- 1) **DATA COLLECTION**
- 2) **DATA ACCESS**: Comprehensive data reflect the totality of care: Who did what, where, when, to whom, and especially why
- 3) **DATA MANAGEMENT**: Editing the data reduces inaccuracies and the effects of creative billing practices; e.g., upcoding, exploding and unbundling. Other data-management techniques that can work within the constraints of the existing claims trail include: recognizing the severity of illness, sorting by diagnosis and diagnostic cluster, re-sorting or re-configuring existing data to enable us to understand the performance of the provider, the specialty, the facility, etc. in an episode-of-care
- 4) **DATA ANALYSIS**: of trends and statistics

⁴ Health care is becoming increasingly unaffordable, but prices vary among communities. Both access to health care and medical practice vary. The type of insurance is related to the cost of care and will determine some of the treatments. Thus, we need to know how the patient's care will be financed; e.g., by Medicaid/Medicare, prepayment, or fee-for-service methods? Finally, is there some cost-sharing insurance device (e.g., insurance agreement where the patient's financial responsibility varies with the choice of service)?

⁵ In terms of developing a promising infrastructure for information technology

⁶ the tools are algorithmic research about appropriateness, small area analysis, evaluations of practice against pre-established patterns of medical diagnostic efforts and treatments; sampling, plus explicit and implicit statistical quality control

- 5) **DATA PRESENTATION:** i.e., medical management needs both standardized and ad-hoc report capability

DATA ACQUISITION, ENHANCEMENT AND AVAILABILITY

Why is the recent progress in data acquisition so important? It is because the two-way, electronic communication about eligibility for coverage of health care and review of procedures (utilization review) is now a reality at the point-of-service. It will not be long before efficient claims adjudication and payment, as well as appropriateness-testing are possible during episodes of health care⁶. Knowledge of the process of health care delivery is a prerequisite. This may be derived from the diagnoses and procedures on claims and clinical encounter forms or from the medical chart⁷. In the middle of the night these data can help the doctor or nurse care for a patient in sudden difficulty. They can also improve our comprehension and help avoid wasteful duplication⁸.

DATA MANAGEMENT, ANALYSIS AND FEEDBACK

We can begin testing clinical effectiveness⁹, even to the point of monitoring efficacy when our understanding is based on standards and tests of appropriateness.

We can close the feedback loop with this information. Providers will learn, confidentially, how they compare with their peers and if they are meeting nationally-based standards on practice guidelines. Enrolled or participating subscribers and employers need similar information so that they will know what is acceptable standard, when quality is compromised by parsimony, how one judges the quality of a provider¹⁰, and the necessity of a procedure measured against the risk.

The managed care industry and insurers should begin to see themselves as gatherer and protector of information. We are the advocates for patients, helping them make enlightened choices for their medical care. We can help the employer by providing prompt, effective care to their employees. Outcome studies are essential to this process, but do not have to be onerous. Indeed, a lot can be gleaned just from the claims trail, especially when case-mix is considered⁷. Providing comparisons of results of one network provider with those of other groups is educational. Finally, we can give physicians, hospitals, other providers and regulators appropriate access to our own developing standards, literature and experience.

We have to be cognizant of the sensitivity of these data and respect the secrets. We must be especially careful to avoid making value-judgments. Our aim is to assist, not threaten the doctor-patient relationship.

TOTAL QUALITY MANAGEMENT

Managed care and those in charge of employee benefits have begun to apply the principles of Continuous Quality Improvement.^{11,12} Total quality management, another name for the same process, depends upon confidence in the physician as provider. It analyzes change in performance and its

7

HealthChex's Peer-A-Med product (Victor, N.Y.) provides a retrospective claims analysis that is case-mix adjusted. Because it is patient-based and longitudinal, it can be regarded as a "soft" episode of care.

continuity in the delivery process of health care. It provides feedback on the quality of performance and cost of diagnosis and treatment. As in many other industries, health care can use strategies that empower us to make appropriate choices.

This point of view becomes a palpable hope as we examine the critical needs, goals and objectives of feedback systems in managed health care:

- to approach data needs , systematically
- to manage data and information
- to build dynamic data bases
- to sample statistically
- to analyze & communicate
- to network
- to understand and improve health care management
- to reduce cost while maintaining or improving quality
- to establish standards
- to define and possibly exceed customer expectations
- to make information available from patients for providers, carriers, payers and policy holders,
- to move from retrospective review to current monitoring as we seek better medical treatment

INFORMATION SECURITY SYSTEMS

Information security systems should be able to restrict access to select individuals to read, store, retrieve, transmit, and share information. Individuals are privileged according to their legitimate need to know. "The system is expected to be able to trace all data access activities and to know where the data came from, who generated and who accesses the data, and when. It permits certain access activities only when specified pre-conditions are met. Inference and aggregation activities must be managed by the system to eliminate potentially illegal or damaging deductions and inferences."¹³

The Hippocratic oath holds the relationship between patient and physician, sacrosanct. To build trust, both parties must have confidence that their privacy will be protected. Patient records often contain sensitive data which must be held private. Their health problems are particularly vulnerable to unwarranted disclosure in a loosely protected data-management environment. Consequently, the basic protocol for handling patient records is rigid. And, in many states and organizations, it is compulsory to have standards and enforcement procedures which protect confidentiality.

WHO IS THE KEEPER OF THE INFORMATION?

Some organizations and individuals are legitimately involved in the health care delivery process or its documentation. Disclosure laws and policies in these instances allow privileged individuals access to the patient-record so that they may perform their duties. The discrete, but ever expanding list of authorized individuals include medical support personnel and others: nurses, aides, mental health/substance abuse personnel, social workers, administrative staff, pharmacists, claims adjudicators, quality assurance staff, legal guardians, researchers. While the physician has the responsibility to manage the patient's care and document his history and innermost concerns, sometimes there is also a requirement to disclose pertinent medical or behavioral situations.

All these people have access, but there must be some way to protect sensitive information. We can see an example of modern approaches to this problem with Lotus Notes¹⁴. Notes uses two forms of encryption to secure confidential or sensitive data. Both use long strings of numbers (digital keys) that work with Notes' software to encipher and decipher information.

- "The dual-key system, with a public key and a private key for each user, is used for electronic mail. Each user has a private key, included in the user's ID, and a public key is included in her entry in the Name & Address Book. When the user seals an E-mail message, it is encrypted with her private key. When the recipient opens the message, Notes retrieves the sender's public key from the Name & Address Book to decrypt it. Successful decryption demonstrates that the message is authentic."
- "The single-key system is used to encrypt documents and fields in documents that stay in the database where they were created. As a result, the complexities of a two-key system aren't necessary. A key can be created by one user and shared via E-mail with others. When a secret key is received, Notes includes it in the recipient's user ID file. The name of a secret key itself is not a secret, but any user who needs it to decrypt a document must have it in his user ID. These keys are for groups who want to share confidential information."

Notes also relies on masking to protect parts of documents.

- "Masking forms can be created to reveal some fields in a document while hiding others. Masks are invoked by form formulas written into the views. This ability to control display of a document in forms other than the one it was created in is a powerful capability, and makes possible a couple of very different approaches to structuring a Notes application [i.e.,] 'One Document, Many Forms.' "

AN APPROACH TO INFORMATION SECURITY¹⁵

T. C. Ting, a major contributor to our understanding information security, emphasizes the role of the user as the major control point in authorized access to confidential data. In other words, those of us managing systems and those of us managing patient care would have an obligation to define access rights based upon the defined user's role. In accordance with this policy, all authorized actions must be traceable to their source and to their end. Since data access rights are often context-dependent and content-specific, the development of appropriate techniques for handling requests for protected information may be a complex task. Tracing data-access activities means being able to identify who generated and accessed which information, where, when and how within the context of the restrictive definition of the user's privilege. An audit trail is employed to monitor the transmission of data both in and out of the system. Finally, it is important that the tracking process be circumspect and not allow the process of privileging itself to invade the privacy.

Personal medical information can be damaging to the individual if it is used irresponsibly. A breach of confidentiality may arise when the data generate fear of the unknown (e.g., the implications of HIV testing for AIDS). Indeed, the responsibility for clearance depends upon the sensitivity of the data. Also, the degree of complexity of the information may have to be a consideration. When we examine the flow of clinical information, we can see the difficulties in protecting it:

In patient care, there is a chain of events which begins at the point of contact (see reference 7) and ends in some clinical outcome. At each stage in the management of a patient, the system must be able to identify which data are requested, how and by whom they are used and which are to be protected. Permission may even be granted by the patient with implied or explicit consent. Nevertheless, this procedure may not be adequate to protect the patient's privacy because of unanticipated factors in the medical interaction. There are no rules to verify who the responsible party is, where and how the data will be stored, or even how able the patient is to make informed decisions under duress.

SECURITY SYSTEMS DESIGN IN TODAY'S CLINICAL ENVIRONMENT

The security and confidentiality of vital clinical information concern us today.

"Not all systems have passwords, and they are often bypassed; only some have tables that define what any particular individual should be able to do; and only a few have audit trails of what has been done. Most platforms are weak. In no present system is security strictly compartmentalized, as at the B1 level⁸. Moreover, the majority of the applications are written in special languages, some of which require their own operating system. Some applications still stem from the 1960's and are written in assembler. Some are written in BASIC, others in MUMPS -- the Massachusetts (General Hospital) Utility Multi-Processing System -- a 1960s system optimized on the variability of medical data, still others in PICK. Independent data base structures have only recently come under consideration. UNIX designs have just begun to appear. There have been reasons for this developmental pathway, and there are reasons, today, for overcoming the deficiencies that have resulted."¹⁶

Physicians are trained to think tentatively and to assign a label to symptoms even before they can make a firm diagnosis. Also, creative billing practices tend to increase both the severity of illness and its reimbursement. This practice can have far-reaching consequences to the individual as can be seen in the following example:

"[I]n children, the diagnosis of asthma can deny insurability or increase insurance rates, sometimes for life. In young children, there are many borderline cases of wheezy bronchitis that can provoke an asthma-like attack that should be treated as if it were asthma." All personnel involved should be alerted to the possibility of a life threatening, asthma attack. Avoiding a word like asthma and talking around the issue can lead to misunderstanding. Also, "such fine distinctions may be lost on those who review or abstract a record after the fact. To label a patient as asthmatic may be equivalent to a pronouncement of guilty without a trial."¹⁷

Confidentiality would be easier to understand and manage if the user list (or table of terminals) were not explicit in terms of privileges. It would be preferable if each user could be profiled at the time of request in relation to the quality and sensitivity of the data. In an object oriented sense, the model would permit categorization by virtue of object type⁹. This method varies category by

⁸ Department of Defense nomenclature

⁹ As defined by unique identifiers, the values for the attributes of the object (i.e., the "state") and the methods that operate on the attributes (i.e., the "behavior")

category, or even item by item. Administratively, one must maintain a file about which user has which rights, when, where, and to what extent. Furthermore, stipulations to override must be precise, and enforced when the requirements are not met.

Needless to say, the documentation (input) procedures that allow user profiling and monitoring, must not be onerous. The exigencies of the care of a sick patient and the resistance of doctors to bureaucratic interference (the "hassle" factor) suggest a user-friendly, yet structured format. We have found that front-end management information systems (MIS) can help.

In our system of managed health care we are learning that different groups have special needs. We have, therefore, decentralized much of our management activity including cost-analysis, quality assessment and security systems. Distributed processing encourages the detail we need for categorization and observation on a local level while it also helps us attain an orderly flow of data (through-put) and relevant information. A relational database of my own design, Medicine Optimally Managed¹⁸, built on a multi-valued software tool, Advanced Revelation, is being explored to facilitate this complex and interactive process.

THE REAL (MEDICAL) WORLD AND RECOMMENDATIONS

The aforementioned are mostly concerns of the computer system. What is privileged and who is authorized must be defined by the medical parties responsible for the confidentiality of the patient encounter and record.

A complete description of the process of securing medical information cannot be given here. The key to the methodology of accessing while also protecting vital data and information is first, to identify the users. Then, we must define their rights and establish their responsibilities. Propriety must be the rule, and exception unacceptable. These are critical everyday operational issues. Medical security systems must not only be reactive, but proactive and interactive as well¹⁹. These systems and the methodological development are a continuous professional challenge, best handled by a scrupulous, systematic team in charge of quality-control.

ADDENDUM (FROM LOTUS NOTES²⁰)

Security is part of the design of any Notes application, and control of access to the application is just as important as the security of other corporate data and computing resources. Notes' tools for tailoring the database include:

- **The Access List.** While everyone in a work-group may have access to a database, that access probably will not extend company wide. The most basic form of security for a Notes application is a list of its users. Each application has its own user access list, created under the File menu in Database User Access Control. The list can name groups or individuals, and assigns each entry one of seven access levels, ranging from no access to Manager access, which grants the authority to delete the entire application and all its contents .
- **User Privileges.** Assigning user privileges makes it possible to control access not only to the database as a whole, but also to particular views and forms within it. This restriction is accomplished by assigning privileges to users and groups named in the Access List, and setting the corresponding privilege for the views and forms they will be allowed to use.
- **Encryption.** Encryption provides a system of access control that works in parallel with user privileges. Data is encrypted on a field-by-field basis in documents, and the decrypting key distributed to a selected group of users. The result is that especially sensitive data can be hidden even from legitimate users of the application. Encryption protects the data in sensitive applications even if the physical security of the server has been breached.

Advanced application development tools and techniques can help Notes developers and administrators deal with the most nontraditional aspect of Notes --- setting up applications for use by groups. These capabilities include using filters, assigning access controls and user privileges, creating encryption keys and encrypted fields and documents, and using masks in a database structure that departs from the simple many-documents-many-forms relationship of previous examples.

An example of a more complex application structure [is] a personnel database that gives different groups access to different information contained in one set of documents. The designer uses database-level security features, such as access controls and encryption keys, to provide each group of users with ready access to what they need to know while protecting sensitive information.

REFERENCES

1. T.C. Ting, "A User-role Based Data Security Approach", in *Database Security: Status and Prospects*, 1988; Ed. C.E. Landwehr, Elsevier Science Pub. B.V., N.-Holland: 187-208.
2. Thomas S. Kuhn, *The Structure of Scientific Revolutions*, 1970; University of Chicago Press, Chicago, IL.
3. Kaplan JG and Brophy J,: "Informed Decision Making -- An Essential Metamorphosis in Health Care." *Medical Interface* 1992;5(5):62-70.
4. Couch, J.B.: *Health Care Quality Management for the 21ST Century*, 1991; Pub. by American College of Physician Executives: 132-3.
[Here the focus is upon "what," not "who," and where our ultimate goal is to meet or exceed customer expectations.]
5. Chassin MR, Kosecoff J, Park RE, Winslow CM, Kahn KL, Marek NJ, Keeley J, Fink A, Solomon DH, and Brook RH. "Does Inappropriate Use Explain Geographic Variations in the Use of Health Care Services? A Study of Three Procedures." *JAMA* 1987; 258:2533-2537.
6. Kaplan, J. "Real Time Management Technology: Improving Accountability, Efficiency and Effectiveness," presented at the Group Health Institute June 12, 1990, Group Health Association of America: 413-424; See also, "Accountability, Efficiency and Effectiveness." *Medical Interface* 1990; 3:13-17 - with references in Volume 7.
7. Weed LL: *Medical Records, Medical Education and Patient Care*, 1970; Chicago, IL., The Press of Case Western Reserve.
8. Kaplan JG, Alvaro N, Auyer P: An Argument for a Point-of-Contact Data Management System. *Medical Interface* 1989;2 (2) 23-28.
9. Kaplan, JG "The Single Pathway Towards Effectiveness." *Medical Interface* 1990;3(10):31-34.

[N.B. The choice of the term "single" was deliberate, referring to least common denominator. We do not wish to imply by "single," "only" or "the only one."]
10. Kaplan JG, Alvaro N "Medical Credentialing and the Quality Assurance/Risk Management Interface." *Medical Interface* 1991;4(3):24-29.
11. Deming WE *Out of the Crisis*, 1986; Cambridge, Mass., Pub. Massachusetts Institute of Technology, Center for Advanced Engineering Study.

12. Juran JM *Managerial Breakthrough*, 1964; New York, N.Y.:McGraw Hill International Book Co.
13. Lincoln TL, Essin D, in a paper "The Computer-Based Patient Record:Issues of Organization, Security, and Confidentiality" International Federation of Information Processing; (IFIP) Working Group 11.3 on DataBase Security; Fifth Working Conference Nov. 3-7, 1991 Shepherdstown, W. Vir.
14. David DeJean, Sally Blanning DeJean *Lotus Notes At Work* Lotus Books, 1991; Pub. Brady: 174, 179, 185-186.
15. T.C. Ting, "A User-role Based Data Security Approach", in *Database Security: Status and Prospects*, Ed. C.E. Landwehr, 1988; Elsevier Science Pub. B.V., North-Holland: 187-208.
16. Lincoln & Essin; ibid
17. Lincoln & Essin; ibid
18. Kaplan, Alvaro, Auyer; ibid.
19. Kaplan JG, Wise J, Gay RH: "Advanced MIS Techniques in Treatment Management." *Medical Interface* 1991;4(6)26-30, 50.
20. David DeJean, Sally Blanning DeJean *Lotus Notes At Work* Lotus Books, 1991; Pub. Brady: 174, 179, 185-186.

Implementing the Message Filter Object-Oriented Security Model without Trusted Subjects

Roshan K. Thomas and Ravi S. Sandhu¹

Center for Secure Information Systems &
Department of Information and Software Systems Engineering
George Mason University
Fairfax, Virginia 22030-4444, USA

Abstract

We propose a new architectural framework and implementation scheme, for the message filter multilevel security model for object-oriented databases. Major complications in implementing the model arise from timing (downward signaling) channels that are intrinsic to the nature of object-oriented computations. This is because object-oriented operations are abstract, and often involve arbitrarily complex write-up actions. The solution to close such channels is best approached at the abstraction level of the computational model (rather than fine-tuning individual systems and their implementation characteristics). A fundamental insight, gained in the course of our research, has been to close these channels by allowing concurrent computations in what is otherwise a logically sequential computation. Our earlier work investigated a kernelized architecture that called for a trusted subject (session manager) to manage a tree of concurrent multilevel computations generated by a user session. In this paper we provide an alternate architecture that eliminates the need for trusted subjects and the associated central coordination and management of concurrent computations.

1 INTRODUCTION

A message filter approach to integrating mandatory security in multilevel object-oriented databases was originally proposed in [3]. The main elements of the model are objects and messages. Security is enforced by a message filter component that controls information flow by mediating message exchanges. The original message filter specification is a step in the right direction in modeling and integrating security in a way natural to the object-oriented paradigm. However, it gives no clue as to how such a specification model can be implemented. This has led the authors to investigate implementation aspects of the message filter model [8, 9].

¹The work of both authors was partially supported by the National Security Agency through contract MDA904-92-92-C-5140. We are indebted to Howard Stainer and Mike Ware for making this work possible.

Although the message filtering actions ensure that mandatory access controls cannot be bypassed, they open up the potential for timing channels. In fact, these channels arise due to the abstract nature of computations in the object-oriented model. A fundamental insight, gained in the course of investigating implementation issues for the message filter model, has been to close such channels by executing an otherwise logically sequential computation, concurrently.

In our further discussions, we deliberately use the term (*downward*) *signaling channel* rather than covert channel. A downward signaling channel is a means of downward information flow which is inherent in the data model and will therefore occur in *every* implementation of the model. A covert channel on the other hand is a property of a specific implementation and not a property of the data model. In other words, even if the data model is free of downward signaling channels, a specific implementation may well contain covert channels due to implementation quirks.

As mentioned before, signaling channels arise in our implementation due to the intrinsic nature of computations in the object-oriented data model. Thus any effort to solve this problem would be futile, unless approached at the abstraction level of the computational model. The solutions we have presented in [8, 9] do not take into account covert channels that are specific to individual implementation and hardware platforms. Rather, the validity and applicability of our solutions rests on the assumption that an ideal trusted computing base (free of covert channels) is available.

A kernelized architecture that was investigated earlier in [8] called for a trusted subject (session manager) to manage and coordinate the concurrent computations initiated by a user session. The session manager has to be trusted so that it can deal with multi-level computations. In this paper we give an alternate architectural and implementation framework that eliminates the need for such trusted subjects. The management and coordination of concurrent computations is no longer centralized, but rather achieved in a distributed (and secure) fashion. The new framework offers obvious advantages. First, it eliminates the need for operating system support for trusted subjects. Secondly, it makes security arguments for our implementation easier.

The rest of this paper is organized as follows: Section 2 gives some background to the message filter model and its evolution. Section 3 presents a reworked architecture without trusted subjects, and further describes how concurrent computations can be coordinated in a distributed fashion. Section 4 discusses some informal proofs and section 5 concludes the paper.

2 BACKGROUND TO THE MESSAGE FILTER MODEL

In this section we give some background to the message filter model and the original implementation of the model with trusted subjects. Our presentation is limited to those aspects relevant to the understanding of the results in this paper. For more comprehensive details on the motivation and evolution of the model the reader should see [3, 8, 9].

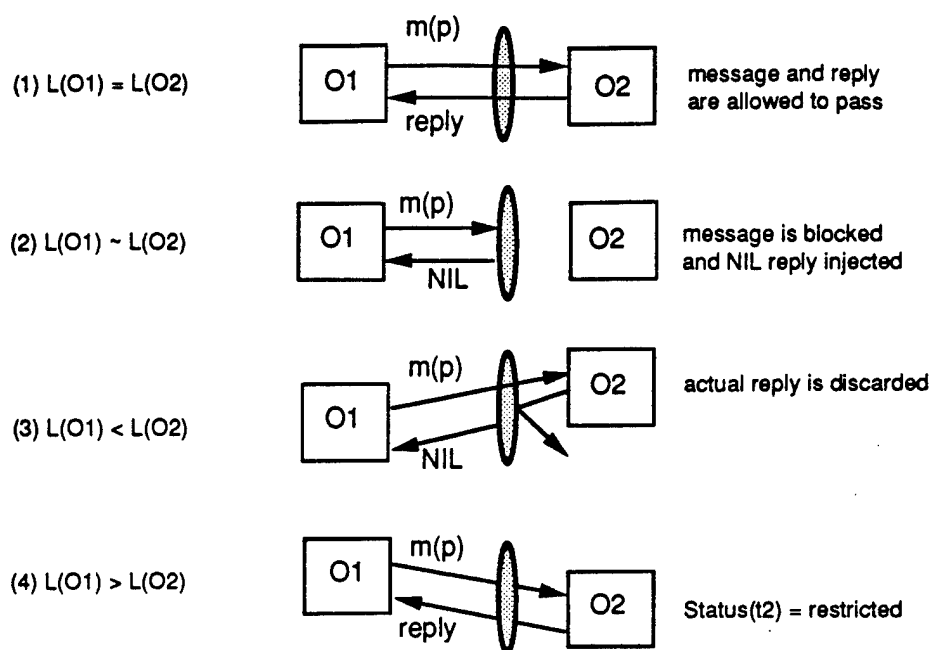


Figure 1: Illustrating message filtering

2.1 The Message Filter Specification

Objects and messages constitute the main entities in the message filter model. Messages are assumed, and required to be, the only means by which objects can communicate and exchange information. Thus the core idea is that information flow can be controlled by mediating the flow of messages. Consequently, even basic object activity such as access to internal attributes, object creation, and invocation of local methods are to be implemented by having an object send messages to itself (we consider such messages to be primitive messages). The message filter takes appropriate action upon intercepting a message and examining the classifications of the sender and receiver of the message. It may let the message pass unaltered or interpose a NIL reply in place of the actual reply; or set the status of method invocations (as restricted or unrestricted). We emphasize that a reply (NIL or other) must always be returned to prevent the sender of a message from blocking indefinitely.

Figure 1 illustrates the message filtering graphically. The full algorithmic specification is given in figure 2.² In case (1), the sender and receiver are at the same security level and the message g_1 and the reply are allowed to pass. In case (2) the levels are incomparable and thus the filter blocks the message from getting to the receiver object and further injects a NIL reply. Case (3) involves a receiver at a higher level than the sender. The message is allowed to pass but the filter discards the actual reply and substitutes a NIL instead. In case (4) the receiver object is at a lower level than the sender and the filter allows both the message and the reply to pass unaltered.

In cases (1), (3), and (4) the method in the receiver object is invoked at a security

²In this and other algorithms, % is a delimiter for comments.


```

% let  $g_1 = (h_1, (p_1, \dots, p_k), r)$  be the message sent from  $o_1$  to  $o_2$ 
% let  $h_1$  be the message name,  $p_1, \dots, p_k$  be the parameters in the message,  $r$  the return value
if  $o_1 \neq o_2 \vee h_1 \notin \{\text{READ}, \text{WRITE}, \text{CREATE}\}$  then case
% i.e.,  $g_1$  is a non-primitive message
(1)  $L(o_1) = L(o_2)$  : % let  $g_1$  pass, let reply pass
      invoke  $t_2$  with  $rlevel(t_2) \leftarrow rlevel(t_1)$ ;
       $r \leftarrow$  reply from  $t_2$ ; return  $r$  to  $t_1$ ;
(2)  $L(o_1) <> L(o_2)$  : % block  $g_1$ , inject NIL reply
       $r \leftarrow \text{NIL}$ ; return  $r$  to  $t_1$ ;
(3)  $L(o_1) < L(o_2)$  : % let  $g_1$  pass, inject NIL reply, ignore actual reply
       $r \leftarrow \text{NIL}$ ; return  $r$  to  $t_1$ ;
      invoke  $t_2$  with  $rlevel(t_2) \leftarrow \text{lub}[L(o_2), rlevel(t_1)]$ ;
      % where lub denotes least upper bound
      discard reply from  $t_2$ ;
(4)  $L(o_1) > L(o_2)$  : % let  $g_1$  pass, let reply pass
      invoke  $t_2$  with  $rlevel(t_2) \leftarrow rlevel(t_1)$ ;
       $r \leftarrow$  reply from  $t_2$ ; return  $r$  to  $t_1$ ;
end case;

if  $o_1 = o_2 \wedge h_1 \in \{\text{READ}, \text{WRITE}, \text{CREATE}\}$  then case
% i.e.,  $g_1$  is a primitive message
% let  $v_i$  be the value that is to be bound to attribute  $a_i$ 
(5)  $g_1 = (\text{READ}, (a_j), r)$  : % allow unconditionally
       $r \leftarrow$  value of  $a_j$ ; return  $r$  to  $t_1$ ;
(6)  $g_1 = (\text{WRITE}, (a_j, v_j), r)$  : % allow if status of  $t_1$  is unrestricted
      if  $rlevel(t_1) = L(o_1)$ 
      then  $[a_j \leftarrow v_j; r \leftarrow \text{SUCCESS}]$ 
      else  $r \leftarrow \text{FAILURE}$ ;
      return  $r$  to  $t_1$ ;
(7)  $g_1 = (\text{CREATE}, (v_1, \dots, v_k, S_j), r)$  : % allow if status of  $t_1$  is unrestricted relative to  $S_j$ 
      if  $rlevel(t_1) \leq S_j$ 
      then  $[\text{CREATE } i \text{ with values } v_1, \dots, v_k \text{ and } L(i) \leftarrow S_j; r \leftarrow i]$ 
      else  $r \leftarrow \text{FAILURE}$ ;
      return  $r$  to  $t_1$ ;
end case;

```

Figure 2: Message filtering algorithm

level given by the variable *rlevel*. The *rlevel* needs to be computed for each receiver method invocation and it is in turn derived from the *rlevel* of the method invocation in the corresponding sender object. The intuitive significance of *rlevel* is that it keeps track of the least upper bound of all objects encountered in a chain of method invocations, going back to the root of the chain. This is required to implement the notion of restricted method invocations so as to prevent write-down violations. To be more precise, we say that a method invocation t_i has a *restricted status* if $rlevel(t_i) > L(o_i)$. The application of restricted invocations is explained below.

The cases (1) through (4) that we have seen so far deal with abstract messages. However abstract messages will eventually result in the invocation of primitive messages. These include READ, WRITE and CREATE³. READ operations always succeed while WRITE and CREATE succeed only if the status of the method invoking the operation is unrestricted. Thus if a message is sent to a receiver object at a lower level (as in case (4)), the resulting method invocation will always be restricted and the corresponding primitive WRITE operation will not succeed. This will ensure that a write-down violation will not occur. Finally, the CREATE operation allows the creation of a new object at or above the *rlevel* of the method invoking the CREATE. The creation of objects lower than *rlevel* is again prevented by restricted invocations.

2.2 Implementation with Trusted Subjects

In our earlier work, we have presented the complications that arise due to downward signaling channels in object-oriented computations [8, 9]. Let us review these briefly. Whenever messages are sent to objects at higher levels, the receiver method should not be able to modulate the timing of the NIL reply. Hence we have no choice but to return the NIL reply immediately, resume execution of the suspended sender, and further execute the receiver object's method concurrently.

Thus the message filter specification calls for an underlying asynchronous implementation/execution model. This could lead to a tree of concurrent computations (methods) as shown in figure 3. Each computation is executed by a separate *message manager* process that implements the message filtering function (in our discussions we often use the terms computations, methods, and message managers interchangeably). Such a tree represents computations forked by a single user (at a single security level) within a single user session. However, each message manager may be executing at a different security level and we thus have a single user but a multilevel tree of computations.

A key feature of an architecture investigated earlier (see figure 5) was the use of a *session manager* process to act as a trusted subject in order to manage and coordinate such a tree. A session manager has to be a trusted subject as it is dealing with computations at different security levels and thus needs to bypass the usual mandatory access controls (particularly the \star property) in a Bell-LaPadula framework. A session manager always maintains a global snapshot of the entire tree of computations as it progresses.

Although conceptually a message sent to a higher level object results in the immediate

³The DELETE operation has not been directly incorporated into the model. It can be viewed as a particularly drastic form of WRITE.

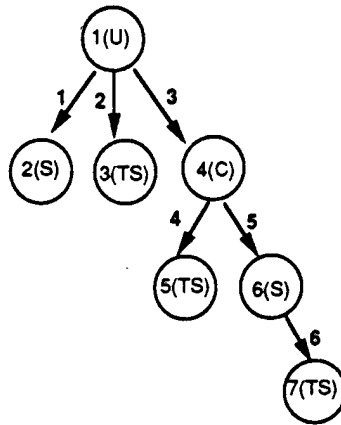


Figure 3: A tree of concurrent message managers

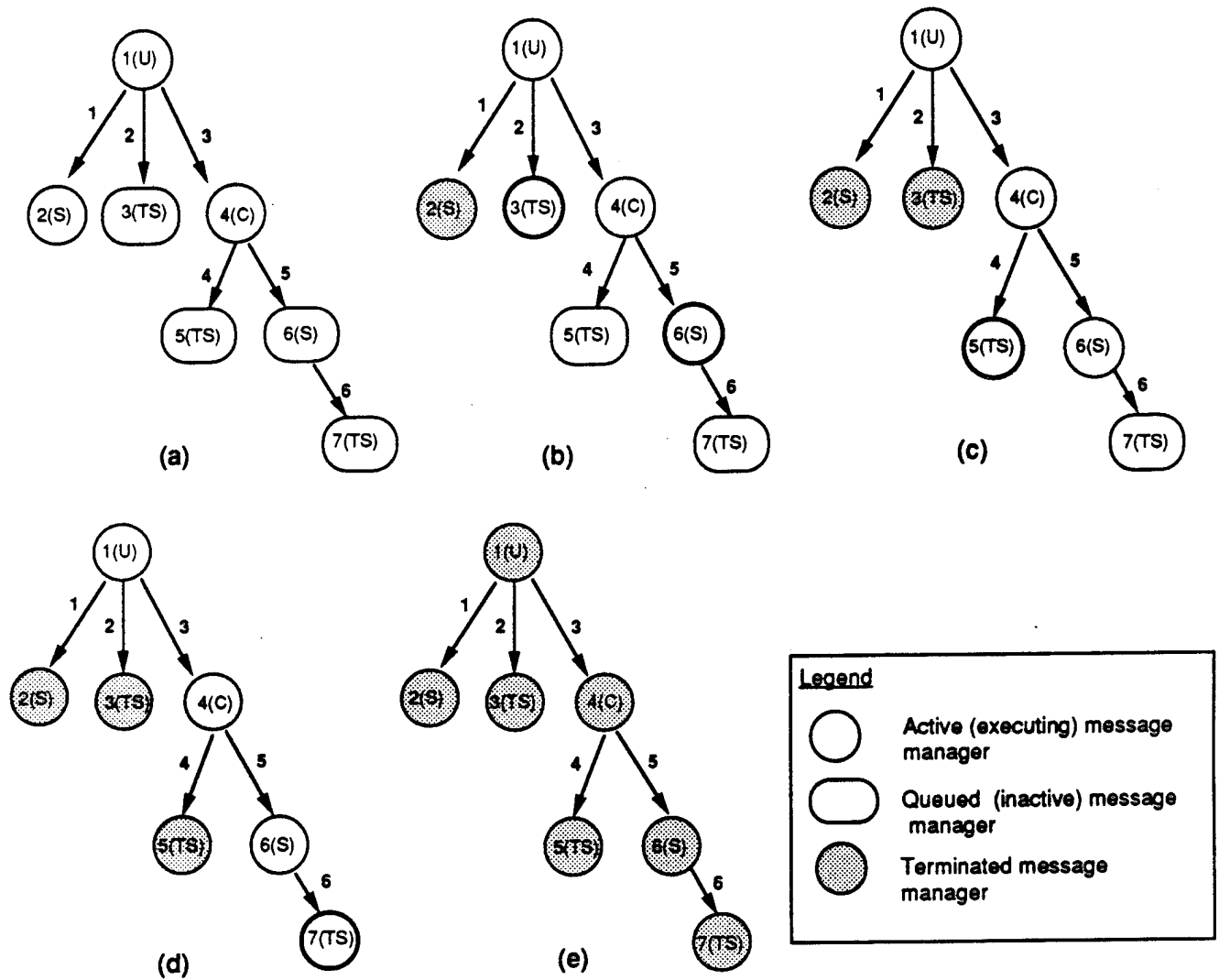


Figure 4: Progressive execution of figure 3

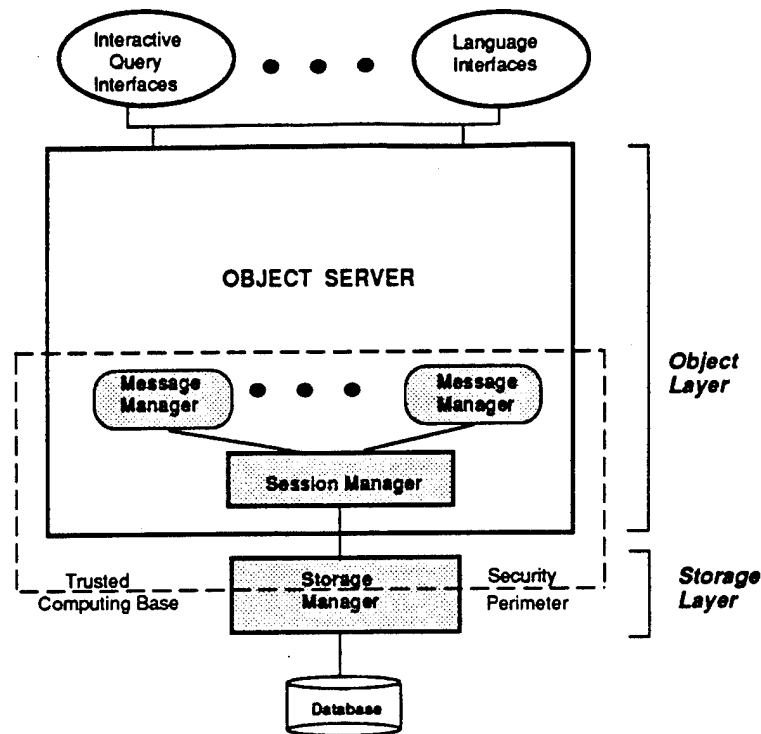


Figure 5: A kernelized architecture with trusted subjects

fork of a new concurrent message manager, the session manager limits the actual degree of concurrency by scheduling computations in a secure and correct manner. Figure 4 illustrates the overall strategy used by the session manager in scheduling these concurrent computations. It utilizes the following invariant in managing a tree of computations:

- **Invariant:** *A computation is started if and only if all the current as well as future computations to the left of it are guaranteed to execute at a higher level or incomparable level.*

Note that this invariant guarantees the following property: for every security level there can exist at most one executing (active) computation at that level at any given time. In other words, some forked computations may be temporarily queued for execution.

The derivation of this invariant is actually motivated by the dual requirements of correctness and security. To see this, we observe that if security were our only objective, we could allow maximum concurrency by enabling computations to unconditionally proceed. However, ensuring correctness (equivalence to the intended logically sequential execution) would then be difficult, if not impossible. Thus the “only if” part of the above invariant is required for correctness. To do this we have to ensure that all writes performed by earlier forked computations at or below the level of a computation say n , are made visible to n (in accordance with sequential precedence). Thus by the time n starts, all these earlier forked computations should have terminated.

The “if” part of the invariant is an artifact of our algorithm and intuitively maximizes the degree of concurrency (as computations are not unnecessarily help up). In fact, we

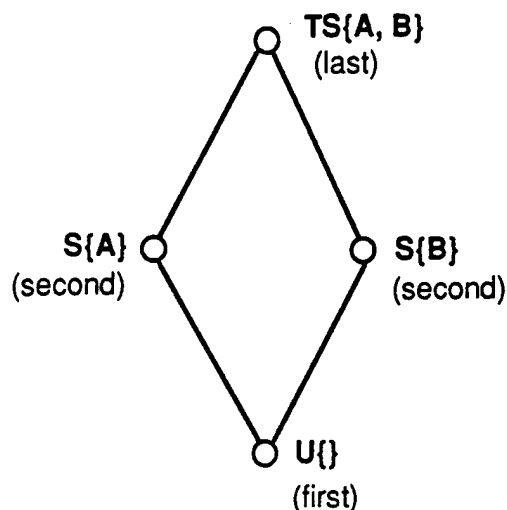


Figure 6: Level by level scheduling in a simple lattice

conjecture that there are many algorithms allowing varying degrees of concurrency.

We illustrate one such algorithm that allows the least concurrency (but guarantees correctness). The basic idea is to follow a level by level scheduling strategy. Thus given a finite set of actions at multiple levels, we first schedule and execute (to completion) all the lowest level actions in the security lattice (one at a time, of course). This is followed by actions at the next higher levels and we continue in this fashion until those at the highest level in the lattice are scheduled last. It follows that whenever there are actions at incomparable levels, they will be executed concurrently (to avoid a sideways signaling channel). For example, given the security lattice in figure 6 we would first schedule and execute all the actions of message managers running at the lowest level unclassified. Upon completion, we would then execute concurrently the actions at the incomparable levels ($S\{A\}$) and ($S\{B\}$). Finally when all actions at both these levels have completed, those at the highest level ($TS\{A,B\}$) are scheduled. Our focus in this paper will be on implementing this simpler level by level scheduling strategy without the use of session managers as trusted subjects.

Now back to our original invariant. The progressive execution of the tree in figure 3 as governed by this “if and only if” invariant is shown in figure 4. The terminated message manager (node) which advances the computations to the next stage is highlighted. Message manager 2 being the first to be forked is allowed to execute immediately. However message manager 3 is queued up. Our invariant guarantees that message manager 3 remains queued (suspended) at least till such time as message manager 2 (and any future children at level top secret) terminate. This action is necessary so that the writes by message manager 2 and its children are made visible to the top secret message manager 3, due to sequential precedence. Message manager 4 at level confidential is allowed to execute immediately on being forked, since all active as well as future message managers to the left of it will be at levels higher than confidential. We also notice that the termination of message manager 2 results in the execution of message managers 3 and 6. In essence, our

invariant guarantees that the execution of a lower level message manager is never delayed due to an earlier forked and executing message manager at a higher level.

2.3 Serial Correctness

As mentioned before, one of the key issues to be tackled with these concurrent computations is related to providing synchronization and ensuring *serial correctness*. We have to ensure that the concurrent execution guarantees the same result (object states) as in the original logically executed sequential (single) computation. A multiversioning scheme for this purpose was presented in [9]. The scheme guarantees that high level methods would read down object states at lower levels that were equivalent to the one in the sequential execution. To enable this, versions identifying lower object states that existed at the time a (higher) message manager was forked are maintained. When the forked high level message manager becomes active, it has the necessary timestamps (in a **local-stamp** table) identifying all versions of objects at lower levels that it will need to process read-down requests. These entries are never modified after a message manager starts. A write stamp (WStamp) at the level of the message manager identifies the next version that will be written at its level. On start, a message manager increments this timestamp entry unconditionally before the first write/update operation and subsequently increments it after every fork request made to the session manager.

3 IMPLEMENTATION WITHOUT TRUSTED SUBJECTS

Having given a background to the trusted subject architecture and implementation framework, we now address the issue of implementing the message filter model without trusted subjects. Thus we can no longer rely on the central control and coordination (of the concurrent computations generated by a user session) that was provided by a session manager. Rather, these computations would have to be managed in a distributed fashion. This also follows from the fact that no system component would at any time ever have a global snapshot of the set of computations in progress. In light of this, is distributed management in a secure manner possible? We assert (and later demonstrate) the feasibility of this based on the following observations:

- The decision to start or queue a computation can only be affected by other computations at a lower level. In other words, one only needs to look down to determine this and thus will not violate mandatory security.
- The termination of a computation (message manager) can result in the subsequent start-up of other computations only at higher levels. This can be accomplished by sending messages upwards and without violating mandatory security or introducing **downward** signaling channels.

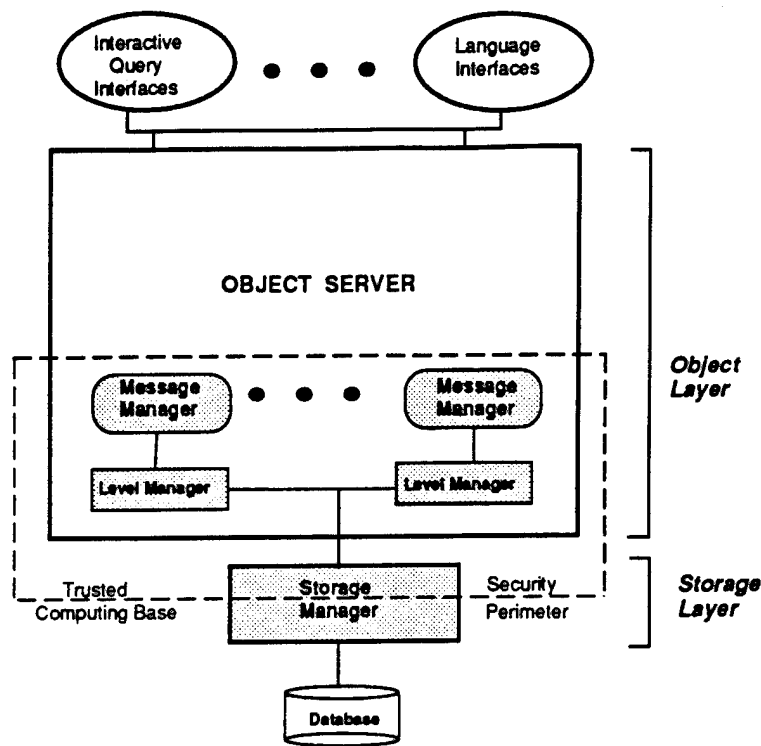


Figure 7: A kernelized architecture without trusted subjects

Figure 7 illustrates the reworked architecture without session managers acting as trusted multilevel subjects. In comparison to the trusted subject architecture in figure 5, a multilevel session manager is now replaced by single-level *level managers*. A level manager is responsible for scheduling and coordinating all computations forked at its level. Message managers are short-lived as they are created and terminated dynamically as the need arises with fork requests. On the other hand, this architecture calls for the existence of a long-lived level manager at every security level for which a computation can be potentially forked.

As can be seen in figure 7, this architecture is a layered one and consists of a storage and an object layer. The security perimeter of the object layer consists of the following primitive operations: SEND, QUIT, READ, WRITE, and CREATE. The READ, WRITE, and CREATE are related to the primitive messages discussed earlier. The SEND and QUIT are system primitives used by message managers (methods) to send (non-primitive) messages and replies. The message manager algorithms for these are shown in figure 8. In our original architecture with trusted subjects, the interface between a message manager and its session manager consisted of two calls: (1) FORK issued by a message manager to its session manager to request the creation of a new message manager and (2) TERMINATE issued by a message manager to its session manager to terminate itself. In our revised architecture without trusted subjects, these calls form the interface between a message manager and its local level manager.

3.2 Achieving Distributed Coordination

We now address the issue of coordinating a tree of concurrent computations to enable overall progress. We begin by describing a few data structures.

Every message manager maintains the following information:

- Local-stamp: a vector of timestamps to process read down requests;
- Fork-stamp: a stamp identifying the message manager's fork order;
- WStamp: the write stamp for versions written by the message manager;

Every level manager maintains the following data structure:

- Current-WStamp: the current timestamp given to objects written;
- Queue: a queue of message managers waiting to be activated;
- Fork-history: a list of ordered pairs (fork-stamp, WStamp);

Our focus in the remaining sections of the paper will be on algorithms to achieve the simpler level-by-level scheduling strategy. Although this scheduling approach is not optimal in terms of the degree of concurrency allowed, it gives valuable insights into how a centralized coordination task can be carried out in a distributed, correct, and secure manner in the multilevel context.

3.2.1 Maintaining Global Fork Order

As mentioned earlier, one of the difficulties in achieving distributed coordination can be attributed to the lack of a global view of the computations as they progress. In particular, without a global data structure such as a tree we would not know the relative order in which message managers are forked (in a sequential execution) by a user session. Knowledge of such ordering is crucial in maintaining serial correctness.

To elaborate on the above, consider the tree in figure 9. With a level-by level scheduling strategy, the fork of message manager 10(TS) by 1(U) will be queued up at the top secret level manager before the fork of 6(TS) by 2(C) (as 2(C) will not be started until 1(U) terminates). However 10(TS) should be dequeued and executed only after the termination of 8(TS) and message manager 8(TS) in turn should be executed only after the termination of 6(TS). This is required to guarantee correctness as the updates of 10(TS) at level top secret should not be visible to either 6(TS) or 8(TS) as they are both to the left of it. On the other hand, we want to make the updates of 6(TS) visible to 8(TS) and the updates of both message managers 6(TS) and 8(TS) in turn visible to 10(TS). Thus there is a need to capture the information that 10(TS) is to the right of 8(TS) and 8(TS) is itself to the right of 6(TS). One could be tempted to obtain the above ordering information by reading off a system low real-time clock, every time a message manager is forked. However, as shown above, forks are not always generated (in real time) in the order consistent with a sequential execution and thus a solution with real-time clocks will not work.

Our proposal here is to derive such an ordering from a hierarchical scheme to generate fork-stamps. Every message manager on being forked is assigned a unique fork-stamp by the parent issuing the fork. The actual fork-stamps generated for the tree in the above


```

procedure SEND(m, p, O1, O2)
if O1  $\neq$  O2  $\vee$  m  $\notin$  {READ, WRITE, CREATE} then case % i.e., m is a non-primitive message
(1) L(O1) = L(O2) : PUSH-STACK(p); t2  $\leftarrow$  select method for O2 based on m; start t2;
(2) L(O1)  $\sim$  L(O2) : WRITE-STACK(NIL); RESUME;
(3) L(O1) < L(O2) : FORK(lmsgmgr, O2, m, p); WStamp  $\leftarrow$  WStamp + 1;
    WRITE-STACK(NIL); RESUME;
(4) L(O1) > L(O2) : PUSH-STACK(p); t2  $\leftarrow$  select method for O2 based on m; start t2;
end case;
if O1 = O2  $\wedge$  m  $\in$  {READ, WRITE, CREATE} then case % i.e., m is a primitive message
(5) m is a READ : if L(O1) = lmsgmgr then v  $\leftarrow$  WSTAMP else v  $\leftarrow$  RSTAMP(L(O1));
    read O1 with max{version: version  $\leq$  v};
(6) m is a WRITE : write O1 with version  $\leftarrow$  WStamp;
(7) m is a CREATE : create O with L(O)  $\leftarrow$  L(O1) and version  $\leftarrow$  WStamp;
end case;
end procedure SEND;

procedure QUIT(r)
    POP-STACK;
    if EMPTY-STACK
    then TERMINATE(lmsgmgr, WStamp)
    else [WRITE-STACK(r); RESUME;]
end procedure QUIT;

```

Figure 8: Message manager algorithms for SEND and QUIT

example is also shown in figure 9. A set of four digits are used for this example with fork requests at levels U, C, S, TS, and TTS. The root message manager 1(U) at level unclassified (U) is given an initial fork-stamp of 0000. It is then required to increment the most significant (leftmost) digit for every fork request issued. Thus 1(U) assigns the fork-stamps 1000, 2000, and 3000 to its children 2(C), 7(S), and 10(TS) respectively. Now a message manager at confidential such as 2(C) is required to increment the second most significant digit of its fork-stamp for every subsequent child it forks. In other words, with increasing levels, a less significant digit is incremented. Hence the top secret message manager in our example will be required to increment the least significant (rightmost) digit (as shown by the fork-stamps assigned to the children of 10(TS)). Thus a message manager in the subtree rooted at 2(C) will always have smaller fork-stamp than one in the subtree rooted at either 7(S) or 10(TS).

In light of our earlier discussion of this example, we now see that 6(TS) will indeed have a lower fork-stamp than 8(TS), and 8(TS) in turn will have one lower than 10(TS). We are thus able to implicitly capture ~~the fork order in these fork-stamps~~. Finally, we note that an appropriate number of digits can be allocated for generating fork-stamps so

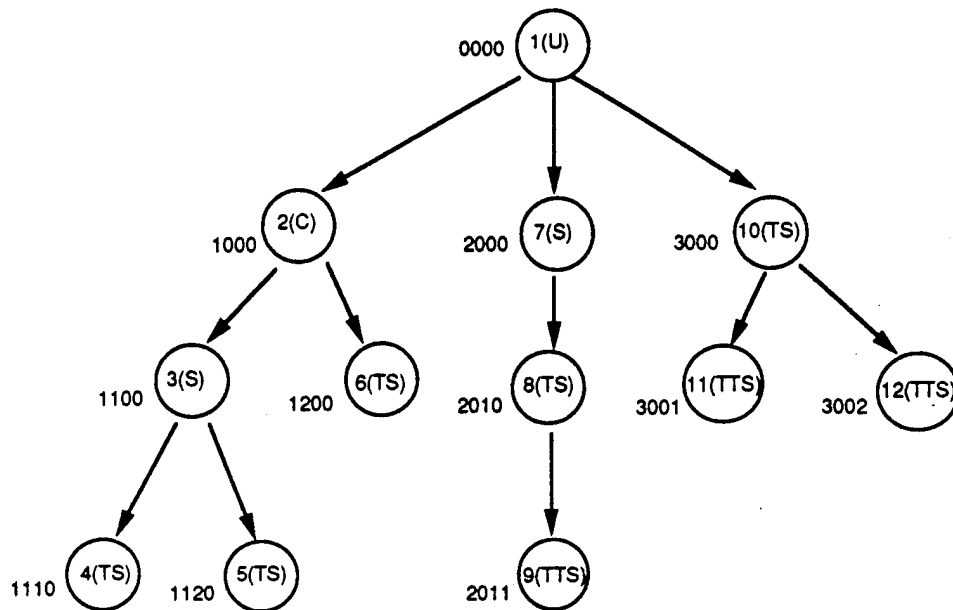


Figure 9: Hierarchical generation of fork-stamps

as to account for the maximum degree of a node as well as the maximum depth of a tree of computations.

3.2.2 The Processing of FORK and TERMINATE requests

A tree of message managers (computations) advances to completion through the generation of FORK and TERMINATE events. Every FORK results in the creation of a new message manager and every TERMINATE could release one or more message managers for execution and subsequent completion. We now discuss the algorithms to coordinate FORK AND TERMINATE events. In our exposition, we will highlight how schemes to guarantee serial correctness are incorporated in these algorithms through the use of multi-versioning techniques.

The algorithm to process FORK requests is shown in figure 10. On receiving a FORK request from a lower level, a level manager creates a new message manager process (computation) and records the fork-stamp (passed on by the parent issuing the FORK) in the message manager's data structure. Now comes the task of initializing the message manager's **local-stamp** table entries. The timestamps for these entries are actually acquired in two phases. The first phase is at FORK time and the second phase is deferred until the message manager starts.

Consider for the moment the first phase. During this phase, a message manager's **local-stamp** entries are initialized for all the levels of its ancestors on the path from the root to itself. The timestamps for these entries are obtained from a vector of timestamps

```

Procedure FORK(O2, m, p)
{
  Create a new message manager mm;

  %Record the fork-stamp passed on by the parent
  mm.fork-stamp  $\leftarrow$  p.fork-stamp

  %Begin phase 1 of acquiring local-stamp entries
  For (every level  $l \leq$  level of the parent of mm)
  do
    initialize mm.local-stamp table entries from p.rstamps;
  End-For

  ENQUEUE(mm);
}
end procedure FORK;

```

Figure 10: Level manager algorithm for FORK

passed on by the parent issuing the FORK request. In fact, every message manager is required to save the timestamps in the vector (rstamps) it receives from its parent and on issuing a FORK, to reconstruct a new vector to give to its child. This newly constructed vector will contain the timestamps from the old vector appended with the write stamp (WStamp) at the level of the issuing message manager (which identifies the latest versions written before the FORK was issued). Obviously, the number of entries in such a vector that is incrementally constructed increases with the number of direct ancestors and the number of security levels (i.e., with the depth of the computation tree).

To see why the timestamps acquired in the first phase preserve serial correctness, consider the path from 1(U) to 5(TS) in the tree in figure 4 (a). In a sequential execution, when 5(TS) is forked the ancestor message managers 4(C) and 1(U) will be blocked (waiting to resume execution). To be more precise, when 4(C) was forked its parent 1(U) was blocked and when 5(TS) was forked 4(C) in turn was blocked. Hence the versions that will be read by 4(C) at the lower level unclassified will be the ones that existed at the time 1(U) was blocked. In a similar fashion, the versions read by 5(TS) will be those that existed at the time 4(C) was blocked. Also, the versions read by 5(TS) at level unclassified will be the same as those read by 4(C) since 1(U) remains blocked until both 5(TS) and 4(C) terminate. Now the timestamps identifying these versions at levels unclassified and confidential are precisely those passed along to 5(TS) when it was forked. Thus equivalence (in read down operations) to a logically sequential execution is achieved. Now for an intermediate level such as secret which is not the level of any of the direct ancestors of 5(TS), the initialization of the secret **local-stamp** entry would have to be delayed until 5(TS) is actually started (for execution). This is thus accomplished only in phase 2. The timestamp for such an entry will identify the latest version written by the last forked message manager at level secret (if any), that is to the left of 5(TS) in the

```

Procedure TERMINATE(lmsgmgr, WStamp)
{
  %Let tt be the message manager that just terminated at level lmsgmgr
  %Let lm be the level manager at level lmsgmgr

  %Update local current write stamp from tt
  lm.Current-Wstamp  $\leftarrow$  Wstamp

  %Update local Terminate-history with the fork-stamp and Wstamp of tt
  Append-terminate-history(fork-stamp, WStamp);

  If queue is not empty
  then
    DEQUEUE(queue, mm);
    START(mm);
  Else
    Send a WAKE-UP message to all immediate higher level managers;
  End-If
}
end procedure TERMINATE;

```

Figure 11: Level manager algorithm for TERMINATE

computation tree. If no message manager was ever forked for the user session at secret, the timestamp for the initial version that existed at the start of the session is used.

Upon completing the first phase of initializing the **local-stamp** entries, the level manager proceeds to queue up the newly created message manager in its local queue. It is important to note that with our level-by-level scheduling strategy, we unconditionally queue up fork requests. This differs from the strategy governed by the “if and only if” invariant presented earlier, where a forked message manager may be immediately started under certain circumstances (thus allowing more concurrency).

The processing of TERMINATE requests is shown in figure 11. When a message manager terminates, the write stamp (WStamp) identifying the latest versions written at its level is recorded by the local level manager. Next the level manager’s **fork-history** data structure is appended with the ordered pair (fork-stamp, WStamp). This captures the fact that a certain message manager was forked in the order given by fork-stamp, and terminated writing versions with timestamp WStamp. Such a history is needed to implement the multiversioning scheme and to guarantee serial correctness. Finally, a level manager dequeues and starts the next message manager (if any) from the local queue. If the queue is found to be empty, a WAKE-UP message is sent to all immediately higher level managers in the security lattice. The receipt of this WAKE-UP message could potentially initiate the execution of queued up message managers at these levels. The processing of these WAKE-UP messages is described in the next subsection.

```

Procedure WAKE-UP
{
  %Proceed if the necessary condition has been met
  If a WAKE-UP message has been received from all lower levels
  then
    If the queue is not empty
    then
      Sort the queue by ascending fork stamps;
      DEQUEUE(queue, nn);
      START(nn);
    else
      Send a WAKE-UP message to all immediate higher levels;
    End-If
  End-If
}
end procedure WAKE-UP;

```

Figure 12: Level manager algorithm for processing WAKE-UP messages

```

Procedure START(nn)
{
  %Let nn represent the message manager to be started
  %Let lm represent the level manager managing nn

  %Complete phase 2 of acquiring local-stamp entries
  For (every level  $l$  lower than the level of  $nn$  for which no timestamp has been obtained so far)
  do
     $nn.local\_stamp[l] \leftarrow mm.fork\_stamp;$ 
    %where mm is the message manager entry in the fork-history at level  $l$ 
    with  $\max\{fork\_stamp: fork\_stamp < nn.fork\_stamp\}$ 
  End-For

  %Update the write stamp (WStamp) from the level manager
   $WStamp \leftarrow lm.Current-WStamp + 1;$ 

  %Begin execution of the message manager nn
  execute(nn);
}
end procedure START;

```

Figure 13: Level manager algorithm for START

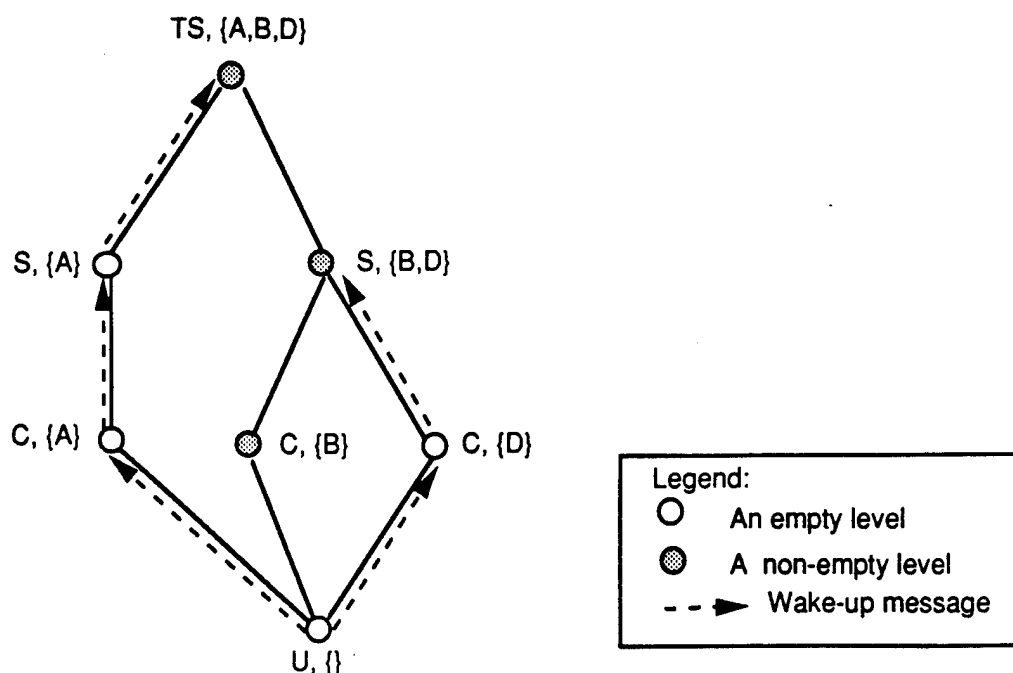


Figure 14: A lattice with WAKE-UP forwarding through empty levels

3.2.3 WAKE-UP Messages and Level by Level Activation

We now describe the semantics of processing WAKE-UP messages and how this achieves the level-by-level activation of computations (see figure 12). When a WAKE-UP message is received, a level manager has to determine if it can release for execution, computations queued up at its level. It can do this only if all activity at all lower levels in the security lattice have ceased. A message manager can be certain of this only when it has received a WAKE-UP message from all immediate lower levels in the security lattice. In other words, this is a necessary (and sufficient) condition for releasing computations at any level. Once this condition is met, a level manager sorts its queue of pending message managers by ascending order of fork-stamps. This is necessary to ensure that message managers are activated in the same order as in a logically sequential execution. After the sort is finished, the message manager at the head of the queue is dequeued and started. Subsequent termination of this and other message managers will cause the queue to be emptied in due time.

If on receiving a WAKE-UP message from all immediate lower levels, a level manager finds its queue to be empty, it simply propagates (or feeds forward) a WAKE-UP message to all immediate higher levels in the lattice (see figure 14 for an illustration). It can do this because it is certain that the queue will remain empty for the rest of the duration of the user session as no more FORK requests will be forthcoming from lower levels.

As illustrated so far, TERMINATE and WAKE-UP requests potentially result in the release and start of queued up message managers (computations). Once dequeued, a common START procedure (see figure 13) is used to complete the second-phase (alluded to earlier) of the task of initializing the local-stamp entries. Now for all lower levels for which no entries were obtained at fork time, the level manager examines the fork

histories. The level manager does this to determine the versions written by the last forked computations that terminated before the fork of the message manager that is to be started. This can be accomplished by comparing the fork-stamps at lower levels to the fork-stamp of the message manager to be started. At each level, the largest such stamp that is less than the stamp of the message manager to be started is picked, and the associated version/timestamp is read. Once the second phase is completed, the level manager provides the **Current-WStamp** value at its level incremented by one, to the message manager. This will enable the just started message manager to write the correct versions of objects at its level. It is important to note the need for incrementing this value by one, for otherwise older versions will initially be overwritten (and this would violate serial correctness).

4 DISCUSSION

Having discussed a level-by-level scheduling strategy, we now briefly and informally argue proofs of correctness, termination, and security. Introducing formal machinery to do this is beyond the scope of this paper and unnecessary as the arguments are simple and straightforward.

- **Correctness.** As in the proof sketches given in [9] we argue serial correctness by showing that the versions read (down) by a message manager are the same as in a sequential execution, and that write operations at its level occur in the same relative order. In phase 1 of the protocol to obtain these timestamps, we have seen how these timestamps identify versions written by blocked ancestors. Since in a sequential execution the ancestors are always blocked due to a running child message manager, the equivalence of these versions follow. In phase 2, forkstamps are used to identify the latest versions written by earlier forked terminated message managers (that are not direct ancestors) at lower levels. Again equivalence follows from the fact that in a sequential execution all lower level message managers that are not direct ancestors of a starting message manager would have terminated. Finally, at every level the message managers are executed in ascending fork-stamp order. Thus the relative order of write operations would be the same in a history generated by our level-by-level scheduling strategy when compared to a second history generated by the logically equivalent sequential execution.
- **Termination.** The proof that with a level-by-level scheduling and execution strategy the entire set of computations will eventually terminate, can be argued from the following: (1) Once a message manager starts, it runs uninterrupted to completion (although the forks it issues may be accumulated for later scheduling). Thus the time needed to empty the queue at any level is bounded; (2) A WAKE-UP message is sent only when the local queue at a level is empty and hence the receipt of a WAKE-UP message is a guarantee that all the computations at the lower sender's level have terminated; (3) There exists no cyclical wait-for relations for WAKE-UP messages among level managers in a security lattice.

- **Security.** This follows from the fact that all subjects are indeed single-level and mandatory access control is never bypassed. The potential for a multi-level trusted subject to leak information no longer exists. Also, we have addressed the issue of signaling channels that are intrinsic to the nature of object-oriented computations.

As mentioned before, we have opted to present the simpler level-by-level scheduling strategy in this paper. But, what does it take to implement the more optimal scheduling strategy governed by the "if and only if" invariant? A FORK request may now immediately result in the start of a new computation. The major complication arises in determining when to release computations at a higher level. When a computation terminates at a level, say *l*, we may have to send WAKE-UP messages to higher levels even when there are pending forked computations at *l* (to allow maximum concurrency). We are currently developing the algorithms formally to achieve this within the architectural framework of single level message managers coordinated by singler-level level managers. These algorithms and associated rigorous formal proofs will be reported in the future.

5 CONCLUSION

In this paper we have reworked a kernelized architecture for implementing the message filter model so as to eliminate the need for trusted subjects. The centralized management of computations now had to be done in a distributed fashion. The new architecture is in line with the true spirit of kernelized approaches to providing security, and would make the message filter model more acceptable to commercial implementation efforts. Having laid the above groundwork, we will be looking into issues involved in supporting multiple user sessions. In particular, we will be investigating the impact of concurrent computations from multiple users on concurrency control and transaction management schemes. Addressing and solving these issues would be critical to the evolution of the message filter model as a full fledged solution for multilevel security needs for object-oriented databases.

References

- [1] W. Kim et al. Features of the ORION object-oriented database system. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [2] D. Fisherman. IRIS: An object-oriented database management system. *ACM Transactions on Office Information Systems*, 5(1):pp. 48-69, January 1987.
- [3] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multi-level security. *Proc. of the 1990 IEEE Symposium on Security and Privacy*, pp. 76-85, May 1990.
- [4] T.F. Keefe and W.T. Tsai. Prototyping the SODA security model. *Proc. 3rd IFIP WG 11.3 Workshop on Database Security*, September 1989.
- [5] T.F. Keefe, W.T. Tsai, and M.B. Thuraisingham. A multilevel security model for object-oriented systems. *Proc. 11th National Computer Security Conference*, pp. 1-9, October 1988.
- [6] D. Maier. Development of an object-oriented DBMS. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 472-482, 1986.
- [7] J.K. Millen and T.F. Lunt. *Secure Knowledge-based Systems*. Technical Report, Computer Science Laboratory, SRI International, August 1989.
- [8] R.S. Sandhu, R. Thomas, and S. Jajodia. A Secure Kernelized Architecture for Multi-level Object-Oriented Databases. *Proc. of the IEEE Computer Security Foundations Workshop IV*, pp. 139-152, June 1991.
- [9] R.S. Sandhu, R. Thomas, and S. Jajodia. Supporting timing-channel free computations in multilevel secure object-oriented databases. *Proc. of the IFIP 11.3 Workshop on Database Security*, Sheperdstown, West Virginia, November 1991.
- [10] M.B. Thuraisingham. A multilevel secure object-oriented data model. *Proc. 12th National Computer Security Conference*, pp. 579-590, October 1989.

Multilevel secure rules: integrating the multilevel secure and active data models

Kenneth Smith and Marianne Winslett

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL 61801 USA
Phone: (217) 244-1263
Email: ksmith@cs.uiuc.edu, winslett@cs.uiuc.edu

Abstract.

Traditional database security is made more complex by the addition of rules to the data model. The security policy must control access privileges and accessibility for rule descriptions, executing rules, and database transitions (events). In this paper we extend the multilevel secure relational model to capture the functionality required of an *active* database, i. e. a database with production rules, able to respond to events. Database rules and events are given explicit security classifications by introducing multilevel secure relations for each. Database rule descriptions are treated as MLS objects. All new user-definable active components (rule actions, trigger detection daemons) conform to mandatory security constraints for subjects. An execution algorithm is given which employs cascading transactions to hide secure rule processing. Implications for implementing the new active functionality in an MLS relational database are also discussed.

1 Introduction

Production rule systems are rapidly being incorporated into database systems, producing *active* databases [16] [5] [9] [11] [12]. Rules extend traditional passive databases with a *responsive* capability, enabling enforcement of user-defined constraints, propagation of the effects of updates, and notification of concerned users about changes deemed relevant. Active database rules both read and alter data, without user intervention, in response to database state transitions (events) and predicates about the database state. Rules can be set-oriented, as in [18], and are thereby able to affect large subsets of the database in one execution. Rules can trigger other rules, initiating long chains of rule executions.

Incorporating rules into the database greatly increases the power of users to act upon the database. Since rules act as a proxy for the user, they can automatically enforce policies and higher level constraints. By contrast, secure databases restrict access to the data objects they manage. The users and processes allowed to retrieve and alter a data item are carefully controlled by the security policy.

The effect of incorporating active rules on database security is just beginning to be explored. Many aspects of rules and their execution have security implications:

- *Rule descriptions.* Rule descriptions are a form of data, and the database must make it possible to restrict access to them. For example, corporate knowledge bases can encapsulate expertise acquired at a large cost. The descriptions of rules in the knowledge base may contain equations and protocols which are classified.
- *Rule triggers.* Rules are triggered by events, transitions in the state of the database. No unclassified rule should be able to see, and trigger on, an event performed by a classified subject. The database must make it possible to classify and protect events as secure objects.
- *Rule actions.* Rules can read and update the database. The database must make it possible to restrict the access privileges of rule actions, as otherwise an unclassified user could use a rule to read classified information, such as a flight destination, or to falsely update it.

Each of these requirements must be met and integrated into a single and comprehensible security framework.

In this paper, we use the multilevel secure relational database model developed in a previous paper [15] as a framework for active database rules. We extend this model with active rules, which we call *multilevel secure database rules*, or simply *MLS rules*, achieving active database functionality without violating mandatory security.

MLS rules follow the general event-condition-action (ECA) rule model [9]. Rules are triggered by an *event*, a *condition* predicate then is evaluated, and, if the condition evaluates to true, an *action* is performed. As described in Section 2.1, MLS rules are both *set-oriented* and SQL-based, as are

the rules of the Starburst DBMS [18], in contrast to general production rule systems found in expert systems.

MLS rules are adapted to suit the MLS environment in several ways.

- Rule descriptions and triggering events are represented in security-labeled tuples, protected MLS objects in the DBMS.
- All user-definable active components (rule actions, trigger detection daemons) are made to conform to mandatory security constraints for MLS subjects.
- A rule implementation strategy to mitigate covert storage channels is described.
- An execution algorithm, called cascading transactions, to mitigate covert timing channels is described.

Although many other database functionalities are affected by the incorporation of rules (concurrency, recovery, efficient access, etc.), we focus on the interactions of rules with MLS security in this paper, and discuss the other functionalities (efficiency and transactions) only as they relate to security issues.

A few previous works have discussed security in systems incorporating various types of rules. Matthew Morgenstern [10] considers the problem of covert inference channels in deductive (expert and database) systems subject to mandatory security. Thomas Berson and Teresa Lunt [1] discuss some research problems to be solved in bringing production rule systems under mandatory security requirements. In [8] Teresa Lunt describes an MLS object-oriented database model with constraints, including a brief description of backward chaining rules. Thomas Garvey and Teresa Lunt [4] extend this work by defining a mandatory security policy for a production rule (or knowledge based) system based on an MLS object-oriented database.

Our work differs from that of [4] because we are concerned with active database rules instead of expert system (artificial intelligence) rules. Specifically, our rules execute in the context of atomic transactions (although they can apply globally). Our rules also incorporate a triggering event which makes them efficient in a database environment and limits them to forward chaining execution (which is suited to active applications). Efficiency of rule execution is vital not only to retain ordinary database functionality, but also to minimize opportunity for new covert timing channels. Set orientation (as described in [18]) makes our rules compatible with the database environment, and also improves efficiency. Additionally, our work differs from that of [4] because we base our view of an MLS database on the semantics described in [15].

In Section 2 we extend the security model of [15] to incorporate rules. Specifically, we describe rule descriptions and their representation as MLS data objects in an MLS *rules relation*. We also describe how rules are activated, deactivated, and represented to the user. In Section 3, we discuss the relationship of our work to two systems which also address both rules and security issues: the Starburst

active rules system, and the proposal for an MLS knowledge-based system by Thomas Garvey and Teresa Lunt. In Section 4 we describe the execution of rules in database transactions; both the simple case of rules which execute at the same level of the transaction which triggered them, and the more difficult case of those which execute at higher levels. In Section 5, we discuss how rules can be used to augment the MLS model itself through examples. In the final section we summarize and describe our future work.

2 Extending the MLS model to Incorporate Rules

In this section we show how the MLS relational model presented in [15] is extended to permit the definition and representation of MLS rules. We define MLS rules sufficiently to give an example illustrating their use, and to introduce our representation for MLS rules, the rules relation.

2.1 A Syntax for MLS Rules

Our MLS rule model borrows heavily from the database rules of Starburst [18] [17]. The following MLS rule syntax is nearly identical to that for Starburst active database rules, except as noted.

An SQL-like syntax is used in MLS rules because it is set-oriented, and is compatible with many database systems. The events, conditions and actions of rules pertain to *sets* of items described by MLS-SQL constructs which are used to express rules. Set-oriented rules are vital (for efficiency) in databases which deal with large amounts of data via set structures such as tables in the relational model and collections in the object-oriented model. For example, a cost of living raise to an entire division can be regarded as a single event. Similarly, a condition may refer to the set of employees receiving raises, and an action may perform a set of corresponding updates to another relation. Set orientation also distinguishes database rules from general expert system rules, which usually are not set-oriented in a manner appropriate for use in databases.

The syntax of an MLS rule is:

```
WHEN transition predicate (the 'event')  
[IF condition predicate] (the 'condition')  
THEN operation block (the 'action')
```

Transition Predicate. A *transition* is a group of operations within a transaction. The user-defined actions of a transaction can form a transition, as can the actions of an operation block of a rule executed within a transaction. Rules may execute at the end of each transition.

A transition predicate is one of UPDATED, INSERTED, and DELETED, followed by the name of a relational table of the database. The transition predicate is true if any of the operations (events) specified occurred to the table during a transition. For example, UPDATED EMPLOYEE refers to an

update to the EMPLOYEE relation. The transition predicate must be true for a rule to be triggered by a transition. If a rule has already been triggered by a transition, but has not yet been considered for execution, it is not triggered a second time.

For MLS rules, the events which satisfy the transition predicate must be visible to the executing rule. We elaborate on this in Section 4.

Condition Predicate. A condition predicate is an arbitrary MLS-SQL SELECT statement. If the result is non-empty, the predicate is satisfied. The entire IF clause can be omitted, in which case the predicate is considered always satisfied.

The SELECT statement may refer to any of four *transition tables*, temporary relations containing the values of the set of tuples affected by the current transition. These four tables are INSERTED, DELETED, NEW UPDATED, and OLD UPDATED, containing tuples affected in the named manner. These tables are modeled as main memory MLS relations, which exist from the beginning of a transition until all rules triggered during that transition have executed, and are discussed in Section 3.1.

The MLS-SQL SELECT statement is defined in [15]. Specifically, the SELECT may contain a BELIEVED BY clause if needed, and is evaluated over the part of the database visible to the rule executing it under mandatory security restrictions.

Operation Block. The operation block is a non-empty sequence of MLS-SQL commands¹, comprising a transition. Arbitrary actions are used in some active models [9], and are permissible in ours as long as mandatory security can be ensured. For example, notifying other users can be modeled as insertion to the user's Mail relation, which ensures mandatory security. Transition tables may be referenced by actions. Limited by mandatory security, rule actions can only affect data at the security level of the executing rule.

2.2 An MLS Rule Example

Throughout this paper, we will refer to the following example. Figure 1 shows an MLS starship tracking relation ST, under a security lattice with three security levels: $U < C < S$. The U level collects, condenses, and updates sensor data about the location of local starships, and uses it for purposes such as scheduling room at ports and maintenance facilities. The C and S levels make use of the U sensor information in a more secure manner.

The first two tuples of ST both belong to the MLS entity with eid *Enterprise U*. The third tuple belong to a second MLS entity with eid *Falcon U*. The first tuple contains the beliefs of level U about the *Enterprise U*. This tuple is the base tuple for the *Enterprise U*, and the S tuple is a non-base tuple (overriding the default beliefs from the first tuple). Level S agrees that *Enterprise U* (the *Enterprise* believed by level U) exists, and agrees with level U regarding all data except its mission, which is believed at the higher level to be security rather than exploration.

¹Commands introduced in this paper, such as rule manipulation commands, can be used under some circumstances as well.

Starship	KC	Nationality	Mission	Destination	Speed	Sector	TC
Enterprise	<i>U</i>	Federation	Exploration	Earth	3.0	Venus	<i>U</i>
Enterprise	<i>U</i>	Federation	Security	Earth	3.0	Venus	<i>S</i>
Falcon	<i>U</i>	Romulan	Mining	Jupiter	2.0	Pluto	<i>U</i>

Figure 1: Starship Tracking Relation ST

Under ongoing hostilities, intelligence (at the *S* security level) has detected the Romulan fighting vessel *Nighthawk* on a moon of Neptune. Any ordinary Romulan ship slowing near Neptune is monitored. The *S* belief is that any slowing ship will be joined by the *Nighthawk* on a military mission. We define an *S* rule Watch-Neptune containing the *S* mission interpretation knowledge and appropriate responses. Watch-Neptune monitors slowing Romulan starships near Neptune. If a slowing ship is detected, a tuple for the *Nighthawk S* is inserted, the mission and destination of the slowing vessel can be updated, and a notification of the event is sent to the central office².

```

WHEN UPDATED ST
IF (SELECT * FROM NEW UPDATED ST
    WHERE Nationality = 'Romulan', Sector = 'Neptune', Speed < 1.0
    BELIEVED BY U)
THEN
    INSERT INTO ST
        VALUES 'Nighthawk/S/Romulan/Combat/null/null/Neptune'
    UPDATE ST
        SET (Mission = 'Combat', Nationality = 'Romulan',
            Sector = 'Neptune', Speed = Speed)
        WHERE Starship in (SELECT STARSHIP FROM NEW UPDATED ST
            WHERE Nationality = 'Romulan' and Sector = 'Neptune'
            and Speed < 1.0
            BELIEVED BY U)
    INSERT INTO Central Office Mail Relation
        VALUES 'slowing Romulan ship detected near Neptune';

```

For rule Watch-Neptune to execute with security classification *S*, the following must occur: First, Watch-Neptune must have been triggered in some transition occurring since its last execution. An update to the MLS relation ST, at any level visible to *S*, triggers Watch-Neptune for the transition

²In our examples, we use a shorthand for SQL syntax.

in which the update occurs. Second, the rule's condition predicate is evaluated by a selection on the NEW UPDATED transition table to determine if any Romulan ships have slowed near Neptune. Only beliefs at level *U*, the level of sensor reports, are tested in the condition due to the 'BELIEVED BY *U*' clause. Third, if the predicate is true, the three actions of this rule are carried out.

The actions of Watch-Neptune only affect data and subjects at level *S*. For example, the *Falcon* *U* updated by the rule (the Romulan ship which slowed near Neptune) is now a two tuple MLS entity, with the updated beliefs contained in a new *Falcon* *U* tuple at the *S* level (the *U* level *Falcon* *U* tuple cannot be changed by an *S* rule).

The state of ST after Watch-Neptune executes at level *S* is shown in Figure 2, reflecting both the triggering sensor update at level *U* and the result of Watch-Neptune's execution at level *S*.

Starship	KC	Nationality	Mission	Destination	Speed	Sector	TC
Enterprise	<i>U</i>	Federation	Exploration	Earth	3.0	Venus	<i>U</i>
Enterprise	<i>U</i>	Federation	Security	Earth	3.0	Venus	<i>S</i>
Falcon	<i>U</i>	Romulan	Mining	Jupiter	0.5	Neptune	<i>U</i>
Falcon	<i>U</i>	Romulan	Combat	null	0.5	Neptune	<i>S</i>
Nighthawk	<i>S</i>	Romulan	Combat	null	null	Neptune	<i>S</i>

Figure 2: ST After Watch-Neptune Executes

2.3 Polyinstantiated Rules

MLS rules are MLS entities, as defined in [15]. As an MLS entity, an MLS rule can exist in multiple security levels simultaneously (be polyinstantiated and share the same eid). Consider the example of another rule pertaining to ST, Tracked-Fast-Ships, which helps maintain a running total of the starships being monitored which are travelling at a speed of 2.0 or greater. Tracked-Fast-Ships is triggered by insertions to ST (any visible insertion event). Its predicate is true if any ship has been inserted during the past transition with a speed field greater than 2.0. The action increments a persistent counter variable³ by the number of fast ships inserted during that transition. Note that rules to monitor deletions and updates would be needed to properly maintain the fast ship count as well; these rules are not included here.

```

WHEN  INSERTED ST
IF    (SELECT * from INSERTED ST WHERE SPEED > 2.0)
THEN  UPDATE FASTSHIPS
      SET (TOTAL = TOTAL +

```

³The counter variable is modeled by the single attribute relation FASTSHIPS.

(SELECT count(*) from INSERTED ST WHERE SPEED > 2.0));

Tracked-Fast-Ships shows the need for a rule to exist (and execute) at multiple levels simultaneously in an MLS environment. At each level, the number of ships seen may be different. For example, if the Nighthawk were found to be a fast ship, the *S* level would see two fast ships (the Nighthawk and the Enterprise) and the *C* and *U* levels would see only one fast ship (the Enterprise). Although a single rule running at the *S* level would have sufficient information to calculate the total fast ships visible to each level, an *S* level rule could not communicate this information down to the lower levels, because user-written rules must obey mandatory security. Therefore, Tracked-Fast-Ships must execute at each level to maintain this total. Tracked-Fast-Ships can execute at multiple levels simultaneously if it is a polyinstantiated MLS rule entity, in the Rules relation described in the following section.

All executing rules pertaining to the *ST* relation (three instances of Tracked-Fast-Ships and one instance of Watch-Neptune) are shown in Figure 3.

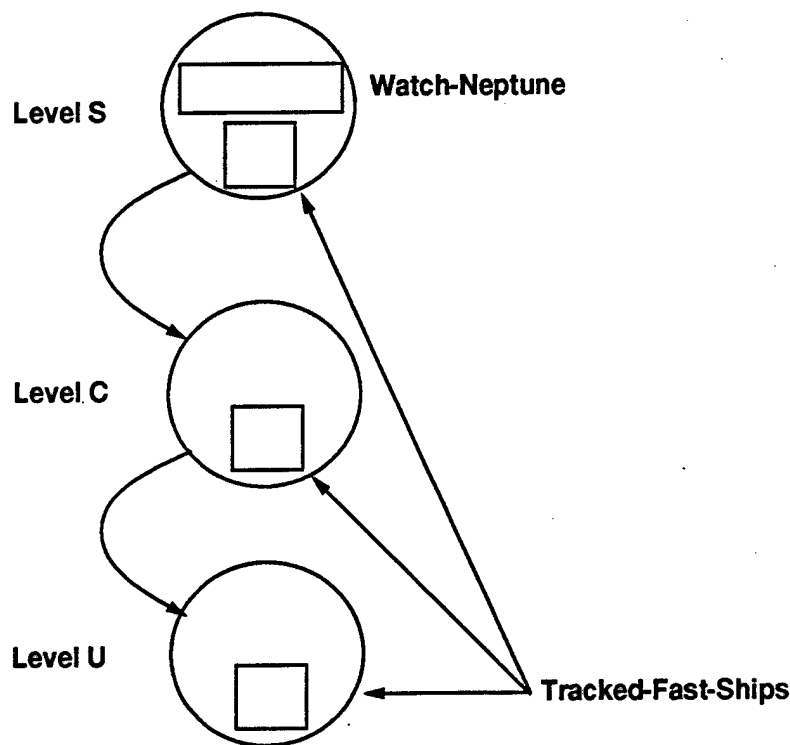


Figure 3: Active Rules for Relation ST

2.4 The Rules Relation

MLS rules are represented in the *Rules relation*, which has eight fields:

- *Name*. User-assigned name of the rule.

- *Transition*. The event triggering the rule.
- *Table*. The table the transition pertains to.
- *KC*. Key class.
- *Condition*. The predicate.
- *Action*. The operation block.
- *State*. The current rule state: descriptive or active ('D' or 'A').
- *TC*. Tuple class.

Name is the name given by the user to the rule. *Transition* and *Table* specify the event which will trigger the rule described. *Transition* is one of {deleted, inserted, updated}; *Table* can be any table in the user's schema, or the Rules relation itself. The four transition tables, described later, are not in the domain of *Table*. *KC* is the level at which this rule was inserted. *Name* | *Transition* | *Table* | *KC* serves as the entity identifier for rules, distinguishing distinct MLS entities in the Rules relation. *Condition* and *Action* are strings containing appropriate text.

The *State* of a rule is either descriptive or active. A rule is in the descriptive state until it is activated by the ACTIVATE command described in Section 2.5, and after it has been deactivated by the DEACTIVATE command of the same section. The rule is only able to trigger and execute while in the active state. The descriptive state permits the representation of nonexecuting rules. The descriptive state of rules is useful because it permits an intermediate stage of rule development, similar to the stage of uncompiled source code in software development.

All operations on the Rules relation follow those described in [15], with the following constraints. Insertion is mediated by a rule parser interface, to translate textual rules into proper Rules relation tuples, and to ensure the values for each field are within their domains. All rules are automatically assigned the descriptive state after being inserted, and can only be deleted in the descriptive state. The beliefs of a level about the predicate and action of a rule can only be updated if those beliefs are not currently activated. Updates to beliefs about rule predicates and actions must also conform to their domains. The *State* field cannot be set in the operations UPDATE, INSERT, DELETE.

Discretionary security, though not explicitly treated in this paper, limits insertions, updates, and deletions to the Rules relation. Only the trusted DBA may perform these actions. However, they may occur at any time during a transaction. Limits are placed on when rules can be activated, however, as defined below.

An example Rules relation is shown in Figure 4, containing MLS rule entities Tracked-Fast-Ships and Watch-Neptune. The predicates and actions of the rules are as defined previously.

Name	Transition	Table	KC	Condition	Action	State	TC
Tracked-Fast-Ships	inserted	ST	U	-predicate1-	-action1-	D	U
Watch-Neptune	inserted	ST	S	-predicate2-	-action2-	D	S

Figure 4: Example Rules Relation

2.5 Rule Activation and Deactivation

When a rule is active (in the active state), the rule is installed in the runtime system of the DBMS. It operates as an MLS subject with a security level equal to *TC*. We follow the Starburst rule implementation model [17], in that installing a rule in the database runtime system involves setting up or modifying a daemon called an *attachment procedure* to watch for events of interest to the rule, and to trigger it at the appropriate times.

To place rules in the active state a subject issues the ACTIVATE command. The form of ACTIVATE is as follows.

```

ACTIVATE
WHERE      P
[BELIEVED BY  L];

```

The ACTIVATE command has the effect of the following MLS-SQL UPDATE command on the Rules relation.

```

UPDATE      Rules
SET         (Predicate = Predicate and Action = Action and State = 'A')
WHERE      P
BELIEVED BY  L

```

P is an MLS-SQL predicate and *L* a list of security levels. In addition to having the effect of this UPDATE, ACTIVATE also installs the activated rule in the DBMS runtime system.

To activate Tracked-Fast-Ships in Figure 4 at level *S*, the following command would be issued by an *S* subject:

```

ACTIVATE
WHERE      Name = Tracked-Fast-Ships
BELIEVED BY  U;

```

Name	Transition	Table	KC	Condition	Action	State	TC
Tracked-Fast-Ships	insert	ST	U	-predicate1-	-action1-	'A'	S
Tracked-Fast-Ships	insert	ST	U	-predicate1-	-action1-	'D'	U
Watch-Neptune	insert	ST	S	-predicate2-	-action2-	'D'	S

Figure 5: Rules Relation After Activate

The resulting Rules relation is shown in Figure 5.

The corresponding DEACTIVATE command is similar.

```
DEACTIVATE
WHERE      P;
```

DEACTIVATE selects a set of active rules at the level of the issuer, removes them from the DBMS runtime system, and sets their *State* field to 'D'.

ACTIVATE and DEACTIVATE are MLS-SQL commands, subject to two integrity conditions (following [17]):

- If a transaction enacts an event (update, delete, insert) on a table, it cannot subsequently activate or deactivate rules triggering on that table.
- If transaction T1 precedes T2 in commit order, rule activations and deactivations by T1 must be visible in T2, and those performed by T2 must not be visible to T1.

The Rules relation also provides a succinct description of the active nature of the database. Such a description is valuable for human and automated reasoning about the characteristics of particular database rule sets [18] [13]. For example, the Rules relation has sufficient information to support an algorithm to detect rule cycles.

3 Related Systems

Examples of rule systems with a security facility are currently quite sparse. In this section, we describe two systems which have also set out to tie rules and security together. We first consider the Starburst DBMS, developed at IBM Almaden [17]. We model our rules and their implementation on Starburst. We describe Starburst in more detail, particularly how rules are processed in transactions, their implementation approach, and Starburst security. We then consider the recent proposal for an MLS knowledge-based system by Thomas Garvey and Teresa Lunt [4].

3.1 Starburst

Starburst is an extensible DBMS which has recently been extended with an integrated production rule system. The rules system has a clearly defined semantics [18], and has been implemented [17] and used in applications such as constraint maintenance [2]. The syntax of Starburst rules is described in Section 2.1.

Rule Execution. Starburst rules execute within the context of traditional database transactions. Starburst rules are triggered throughout a transaction, but execution is deferred until the transaction is ready to commit⁴. As a transaction progresses, therefore, events occur which trigger rules. Triggered rules are placed in a set of *pending rules* until the commit point of the transaction, at which time the *rule processing phase* is entered.

At the onset of the rule processing phase, a rule is picked from the pending rules using a conflict resolution strategy, such as a user-defined rule priority. The rule predicate is then evaluated against transition tables (described below) and the state of the database.

When the second and subsequent rules execute, prior rule executions in the transaction may have already updated the state of the database and, importantly, generated new events and transitions. To ensure predicates are evaluated against the *current* state of the database, transitions generated by prior rule executions are incrementally incorporated into the transition tables prior to the execution of each new rule in the rule processing phase.

If the predicate of the selected rule activation is true, the action of the rule is then performed. Since rule actions can generate further database transitions, the execution of rules may trigger other rules, causing forward rule chaining and augmenting the set of pending rules. These are also executed until no more rules are triggered, at which time the transaction finally commits.

Implementation. Every Starburst rule is triggered by a particular event on a particular table, such as UPDATED ST. (We will call this table-event combination simply an 'event'). Conversely, for every event, there is a set of zero or more rules which have an interest in (may be triggered by) that event. During rule definition time, the interest of a rule in its event is recorded in a persistent *information code* associated with that event. During execution, the information code is consulted to determine which, if any, rules need to be notified about the occurrence of an event. When rule definitions are updated, the information codes are updated as well.

Starburst permits the definition of *attachment procedures*, which can be automatically called after each tuple level operation; transactions gain access to information about events at runtime via attachment procedures. An attachment procedure is associated with a particular event. During a transaction, when an event occurs (for which at least one rule has an interest) the attachment procedure for that event is called. The procedure consults the corresponding information code, and records

⁴Database rules may also be executed the instant an event occurs, or *asynchronously*. These approaches are not taken by Starburst (or us), but have situations in which they are appropriate [9].

the information needed for triggering and for rule execution in data structures local to the transaction. Rules triggered by events thus far (pending rules) are recorded in the *rule processing information* data structure, and a record of the tuples affected by each event is kept in the *transition log* data structure.

At the onset of rule execution, a rule is selected from the set of pending rules for execution, as mentioned above. Starburst permits users to specify that certain rules should be executed before or after other rules. This is vital to give control over apparent non-determinism in rule execution. Rule precedence information is represented in a *global rule information* data structure containing the transitive closure of all rule precedences.

The predicate of the chosen rule may contain a reference to tuples affected by the event which triggered the rule. For example, if the rule triggers on a deletion to table *t*, then the rule can reference the set of tuples deleted from *t* during the current transition. The reference is made to a *transition table* for *t*. Four different transition tables can be formed from the events of a transition on *t*: INSERTED, DELETED, NEW UPDATED, and OLD UPDATED, containing tuples affected in the named manner. Transition tables are treated in the rule as if they were actual relational tables, with a scheme exactly the same as *t*. However, they are implemented by producing them on demand (when referenced in a rule predicate) by searching the transition log data structure of the transaction. The tuples relevant to the table, triggering operation, and transition during which the rule was triggered are selected to produce the table. Note that the information code is specific to the predicates of the rules interested in its event. *Only* the tuples relevant to interested rules are passed to the transition log during a transition.

If the predicate is satisfied, the rule action block is executed in the normal manner. At the end of the transaction, all local data structures (transition log and rule processing information) cease to exist.

Rollback. Starburst permits both total and partial rollback of transactions. Partial rollback resets the state of the transaction to a user-defined checkpoint, and total rollback removes all effects of the transaction.

Rollback is handled by Starburst for all disk-based data structures. However, the Starburst rule manager employs three memory resident data structures which must be rolled back by the rules system when necessary. These three are the transition log, the global rule information, and the rule processing information.

A transition log is maintained in the address space of each transaction, and exists for the entire duration of the transaction. As mentioned, it contains a record of the events which have occurred in the transaction. In the case of a total rollback, the transition log is simply set to empty. In the case of a partial rollback, the events which have occurred since the last checkpoint are removed. This is accomplished by comparing the timestamp of the last checkpoint with the timestamp of each event, and removing later events.

The rule processing information exists in the address space of each transaction during the rule

execution phase. It contains information about currently pending rules. In the case of a total rollback, the rule processing information is set to empty, since no rules are pending. In the rule execution phase of a transaction, partial rollbacks are only permitted within the action portion of a single rule. That is, partial rollbacks may undo at most a subset of the action of one rule. However, the rule processing information is only updated *after* the execution of a rule action, so partial rollbacks never affect the rule processing information.

The global rule information exists in memory at all times, but separate from the address space of any single transaction. It contains (mostly static) rule definition information brought into memory from the rule catalog on disk, to speed access to this information. Global rule information is only changed when rules are redefined, so the only rollbacks which pertain to the global rule information are rollbacks of transactions which modify rules. To undo rule modifications in the event of a rollback, each time a rule is modified, a procedure is queued in the *commit queue* (a queue of procedures which are executed at commit or rollback). This procedure can undo the rule modification action if the *rollback* flag is set when it is executed.

Security. Security in the Starburst rules system is discretionary. Individual users are assigned privileges for the different objects in Starburst (including, perhaps, none at all). Possessors of privileges may grant privileges to other users. Starburst provides privilege types specific to object types. For example, three types of privileges may be assigned to rules: *control*, *alter*, and *activate/deactivate*; each subsumes its successors.

To attach a rule to a table, a user must have both *attach* and *read* privileges for that table. Additionally, the condition and actions of the rule must not conflict with the current privileges of the user adding the rule. To drop a rule from a table, the user must have either *control* privilege on the table or *attach* privilege on the table and *control* privilege on the rule. To alter a rule, *alter* privilege is required. To activate/deactivate a rule⁵, *activate/deactivate* privilege is required. No special protection for reading a rule's definition has been implemented.

Under discretionary security, users acquire and transfer privileges, whereas in MLS, all users and objects are assigned a security label which automatically limits their privileges. MLS currently offers less power to define special types of privileges than discretionary security does, since only two types of MLS access (update and read) have been defined so far, whereas many may be defined in discretionary security. However, MLS is more comprehensive in that every user and object must be consciously assigned to a security class, and in that security extends beyond the standard database access types to covert channels as well, such as denial-of-service leaks, which discretionary security does not consider. Additionally, MLS has a well-defined semantics, as defined in [15], which enables semantic security concepts such as MLS entities to be incorporated into the data model and its operations.

In practice, the two approaches can be used together. MLS systems often utilize discretionary

⁵Deactivation is used in Starburst with a slightly different meaning than we do in Section 2.5; deactivation leaves a rule installed in the runtime system without permitting it to trigger.

security within each access class [3]. This provides label-based security boundaries for larger groupings through MLS, and the ability to define more finely tuned access policies (such as need-to-know) within each MLS access class. The Starburst discretionary security system can, with some modifications, be used within the MLS framework.

3.2 An MLS KBMS

In [4] Thomas Garvey and Teresa Lunt present a proposal for an MLS knowledge based system, which includes a discussion of a rule system. This work is an extension of prior work in MLS object-oriented database systems [8] to address the requirements of knowledge based systems.

The rules described are from an expert-systems model, in which each rule has an 'antecedent' and a 'hypothesis'. Rules can chain in a forward or backward direction. The 'rule object' has slots for its antecedent and hypothesis. An example of inference is given in which users at higher levels make better inferences due to the presence of more information and (presumably better) rules. Conflict resolution strategies are presented when two logically contradictory statements are encountered.

Rule descriptions can be tailored at higher levels, similar to what can be done by updating rules in the rules relation, although the concept of a descriptive (non-active) rule state is not described.

Database-oriented rule features such as a set-oriented rule syntax and semantics, database-specific optimizations such as efficient triggering, and execution in terms of atomic transactions are not described in [4]. For these reasons, the rule model in this paper follows the Starburst approach much more closely than the Garvey-Lunt proposal; our goal is a database-integrated rule model.

4 MLS Rule Execution

In this section we present further extensions to the MLS relational model which enable the activated rules to execute. Our rule execution model is based on the Starburst execution model. We present our model by discussing adjustments to the Starburst model from three perspectives: MLS read and update access restrictions, preventing covert storage channels, and preventing covert timing channels.

4.1 Rule Execution Under MLS Constraints

MLS rule activations execute within the context of traditional database transactions, as do the Starburst rules described in Section 3.1. However, each phase of MLS rule execution must respect the constraints of MLS standard read and update access restrictions.

Events are labeled with the classification of the MLS subject initiating them, and are treated as secure data objects in the DBMS. A rule activation only triggers if it can 'see' the event under MLS constraints. For example, if an S level user performs an update to table ST , the event UPDATE ST is treated (by the user, and internal to the DBMS) as an MLS object with security level S . A U

activation triggering on the event UPDATE *ST* would not see (and be triggered by) this event, but an *S* activation of the same rule would see and be triggered by this event.

The predicates of MLS rule activations likewise respect MLS constraints. Since the SELECT clause used to express the predicate is an MLS-SQL statement, it only selects tuples with an access class dominated by the rule activation executing it. Within the SELECT statement, transition tables may be referenced. We assume transition tables are modeled as (main memory) MLS relations for use in the SELECT statement. Since the table to which the rule is attached (triggers on) is an MLS relation, the Starburst principle that transition tables share a scheme with the attached table remains true for MLS rules.

Execution of the operation block also adheres to MLS access restrictions. Each operation in the block is an MLS-SQL statement, therefore operations only make changes at the classification level of the activation, and data is only read from dominated levels.

4.2 Covert Storage Channels in Rule Execution

The above adjustments to the Starburst execution algorithm prevent reads and updates to MLS relations by rule activations which would violate MLS constraints. However, *other* (internal) data structures of the DBMS can contain secure data as well; especially of concern are the structures which support the definition and execution of rules. If these data structures are accessible to users in a manner violating MLS access restrictions, covert storage channels are introduced. In what follows, we illustrate some of the issues in a plausible MLS architecture, without doing an exhaustive analysis.

Our model does not specify a particular implementation or internal data structures; at the current stage of the development of active database technology, not specifying internal data structures permits a useful degree of flexibility. However, for the purposes of analyzing and addressing the risk of covert storage channels, we assume a Starburst-like architecture supports our MLS rules system. For the purposes of this discussion, we only discuss the covert storage channel issues specific to an active DBMS. Others have set out to implement MLS relational databases [19] [7]; we assume general database issues, such as disk buffer access and transaction management, are being or will be addressed in those systems.

Two major options have been taken in MLS aspects of database implementation, depending on the approach to *trusted subjects*. A trusted subject is a process operating within the database that is permitted to communicate with subjects having a lower security level than its own. These subjects are trusted to not violate MLS restrictions by passing high level information (to which they have access) to lower level subjects.

The TRW [19] architecture permits trusted subjects, while the SeaView [7] architecture does not. We discuss **covert storage channels** from the former perspective, since the mapping to the Starburst architecture is quite direct. Prohibition of trusted subjects is usually associated with a distributed

architecture (with one DBMS running per security level). A discussion from the latter perspective would involve redesign of active features of the runtime system to be distributed. For instance, activation procedures would need to span DBMS's. Since the Starburst rules system is not currently implemented in a distributed manner, our redesign and analysis would be much more speculative in this case.

In the straightforward approach to implementing a Starburst-like active architecture, the rule catalog, information codes, transition logs, rule processing information, and the global rule information data structures could all contain data from any security level. Covert storage channels could therefore be set up through any of these data structures by procedures that read and write them. Where other means cannot be used, trusted subjects must mediate access to these data structures within the DBMS runtime system to ensure storage channels are not set up.

Rule Catalog. The rule catalog is easily brought within the sphere of MLS access restrictions. Its functionality is captured by the rules relation in our model. Since the rules relation is an MLS relation, all access restrictions automatically apply.

Information Codes. Each information code contains a persistent list of all rules interested in an event. Only trusted subjects (no user transactions, rule activations, or other untrusted processes) should be able to communicate information about the information codes within the DBMS. Specifically, this includes the attachment procedures which read the information codes, and the procedure updating information codes.

Global Rule Information. The global rule information keeps information from the information codes and rule catalog in memory, to ensure quick execution of rules. Since it is designed to be widely accessible, and contains information about all rules (potentially from all security levels), it has a high potential for covert storage channels.

One possible solution is to store the global rule information in the address space of a trusted subject which acts as a server to transactions requesting rule information. Another possibility is to divide the global rule information by security level, storing the information for rules at a transaction's security level within that transaction's own address space.

Rule Processing Information. The rule processing information data structure contains a list of pending rules, and a local copy of global rule information relevant to these rules in the address space of the transaction. As illustrated in Section 4.3, a transaction can trigger rules which cannot execute within that transaction, because they execute at a level which dominates the transaction. The pending rules list must only contain rules which can execute within this transaction. Information about triggered rules is passed to the rule processing information by the rule execution module, which must therefore be a trusted subject.

Transition Log. The transition log is implemented in Starburst as a double hash table within the address space of the transaction. It serves to produce the transition tables, and receives input from the attachment procedures. Attachment procedures must therefore be trusted subjects.

Note that transition logs only contains the events which are relevant to rules interested in that transaction. For a *U* transaction, the transition log cannot contain *U* events of the transaction which are only of interest to a *C* transaction (but not to the *U* transaction), even though the events have a security level *U*.

4.3 Covert Timing Channels in Rule Execution

During the rule execution phase, *rule chaining* can occur. That is, a rule's action may generate events which trigger more rules. Rule chaining may continue through many rules before it ends. In an MLS database, rule chaining is limited by the \star -property, preventing it from propagating downward through levels. For example, it is impossible for a rule executing at level *C* to cause a rule in any level below *C* to trigger, since the *C* level events caused by the *C* rule are invisible to rules at lower levels.

We require chaining to progress in a monotonically increasing direction through security levels, *cascading* upward as shown in Figure 6. Thus, all pending rules at level *U* finish before any rules at

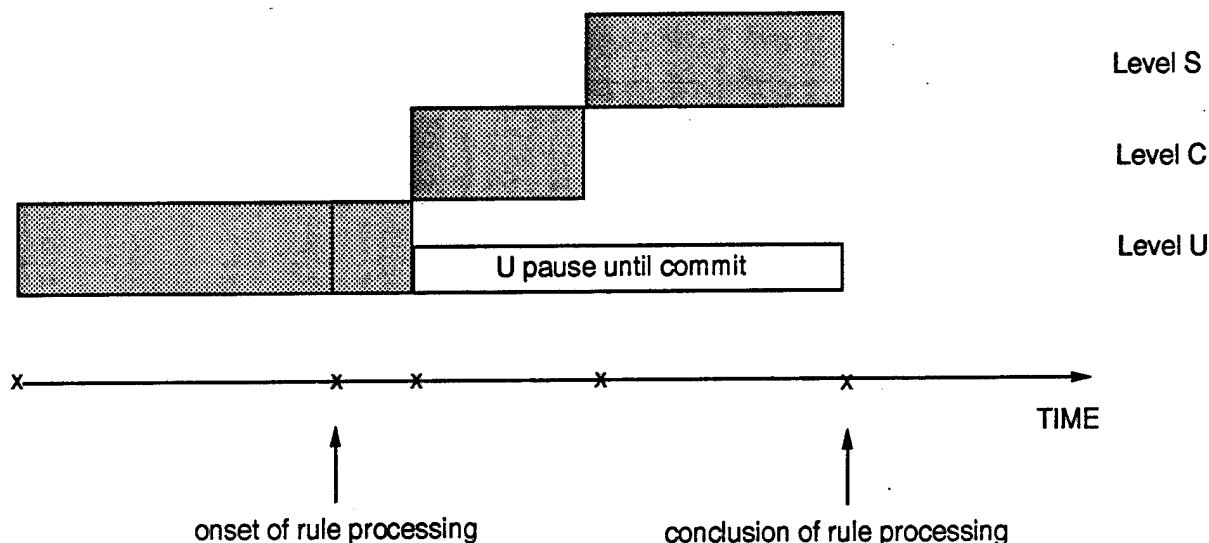


Figure 6: Cascading Rule Chaining

level *C* are permitted to execute. Any *C* rules triggered by the execution of *U* rules become pending rules until execution begins at the *C* level.

In general, the execution of a level's rules is never affected by deferring the execution of higher rules. Low-to-high execution ordering is desirable from a timing channel perspective, since no pause will be experienced while high level rules (like Watch-Neptune in our example) execute between low level rules. Also, if a rollback occurs, no actions at a level higher than the rule performing the rollback have occurred.

It is possible that a timing channel can be introduced by cascading chaining within a single transaction, as shown in Figure 6. While *C* and *S* rules execute, the *U* subject pauses until the transaction

commit message is received; this pause could open a covert timing channel. A second problem exists with implementing cascading chaining within a single transaction as well; the list of pending rules in the rule processing information data structure of that transaction must contain rules from levels higher than the original transaction. Since these rules would exist within the address space of the original transaction, this would open a potential storage channel.

For the above reasons, cascading rule chaining is accomplished in separate, but related, *cascading transactions*. A set of cascading transactions consists of an initial transaction and a set of rule processing transactions at higher levels which perform cascading rule chaining begun by the initial transaction. We illustrate cascading transactions with the following example in Figure 7. During

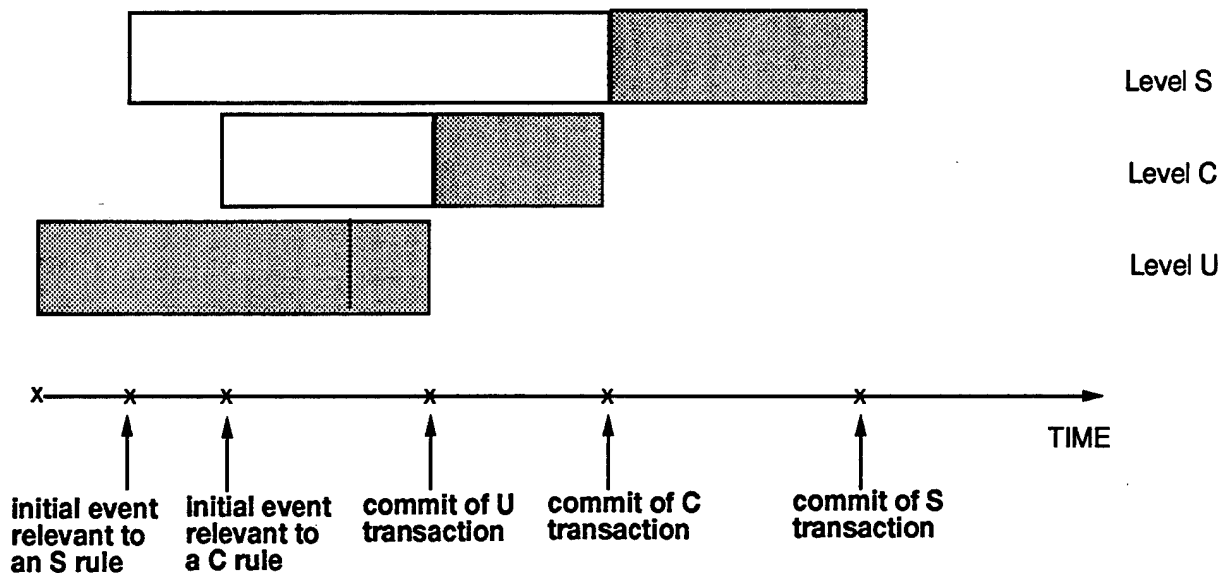


Figure 7: Cascading Transactions

execution in a *U* transaction, an event occurs to a table to which an interested *C* rule is attached. The attachment procedure for the *C* rule normally responds by writing the relevant information to the transaction's transition log. Instead, a new *C* transaction with a separate address space is spawned and receives this information⁶. If the information is relevant to the original *U* transaction, it is also written there (independently). The unshaded region of the *C* and *S* transactions in Figure 7 represents the period during which the transition log and rule processing information of each transaction are being filled. No locks are held and no events occur during the unshaded period. The shaded regions of the *C* and *S* transactions correspond to rule processing.

At the end of rule processing in the *U* transaction, *U* commits⁷ and *C* rule processing begins in the

⁶Additionally, when the *C* rule is triggered, this information is passed to the rule processing information of the new transaction.

⁷At commit, we assume all locks are relinquished and the address space (and data structures contained in it) is reclaimed in a secure manner.

cascaded C transaction. In general, cascaded transactions execute in ascending lattice order. Transactions for parallel security classes may execute in parallel. Any successor of transactions executing in parallel waits until all transactions it dominates are finished before beginning execution. We assume a (trusted) transaction scheduler schedules cascaded transactions.

At the conclusion of rule processing, each cascading transaction drops all its read and write locks⁸. The transaction scheduler, however, passes these locks to the next cascading transaction(s). In this process, all write locks are downgraded to shared read locks. When the final cascading transaction commits, all locks are dropped.

The effect of a set of cascading transactions, considered as a whole, is serializable. When the cascading transactions are considered as a single transaction, the above locking protocol does not adhere to two phase locking since write locks are relinquished before all locks are acquired. However, an early dropping of write locks is equivalent to holding them to the very end, as in two phase locking. Each cascading transaction updates a disjoint portion of the database (due to the \star -property), so it would not matter whether a write lock is held beyond the transaction which used it, or not. For example, a write lock on a U item need not be held until the end of the cascaded S transaction, because level U data cannot be written to by any other cascaded transaction.

The continuing execution of the cascading transactions spawned by a transaction t will be undetectable from t 's perspective, preventing timing channels, with the exception of conflicts with the read locks retained by cascading transactions. A high transaction could attempt to introduce a timing channel by delaying a long time while a low transaction begins a new transaction and attempts to write to an item it wrote to in the earlier transaction. The write lock will be denied⁹ until the cascaded transactions finish.

Usually, it is not possible to tell from the lock manager with whom a lock conflict has occurred, or how long an action was delayed before a lock was granted. Even if it were possible to infer the object of a conflict or a delay length, we believe that the resulting covert channel would have very low bandwidth. To help with this problem, the transaction scheduler can halt a cascaded transaction after a reasonable time-out period. Such transactions could be restarted later, assuming the transition log and rule processing information is saved (higher cascaded transactions which have not executed yet would also have to be saved). Such restarted transactions could suffer from a transition log which is outdated with respect to the database state. To help retain consistency, rules can be written so they rely heavily on transition tables (not on the database state which may be outdated).

⁸We assume individual transactions adhere to the two phase locking protocol.

⁹Unless a form of preemptive locking is used, and the write lock is granted forcing the high transaction to starve.

5 Utilizing Rules in Policies

In this section, we illustrate the usefulness of MLS rules for enacting *policies*. A policy prescribes user-defined responses to specific circumstances which may occur in the course of database processing. The *need to know* constraints of discretionary security and the archival procedures for old data can be expressed as case by case responsive policies. We give an example of a policy concerning the MLS relational model described in [15]. In our example, we consider the need to automatically *coalesce* two MLS entities due to *attribute polyinstantiation* [6].

Under some circumstances, a user will decide two MLS entities in fact represent the same real world entity, and wish to coalesce them into a single MLS entity. The circumstances under which this decision is made may vary by the application and by the specific user, and the policy should not be written into the code of the DBMS. However, it is excessive to require a user to perpetually monitor the database in order to make policy decisions, and to manually perform the coalesce. An external application program could be run periodically to merge entities, but during the gaps between its runs, the user's policy on coalescing will not be enforced.

The flexible and proactive nature of rules enables users to make decisions about how to handle situations which are application-specific or are not fully specified by the data model. These policy decisions are then enacted on the user's behalf by active rules when the situations occur, without the user's presence. MLS rules respond immediately, and adhere to MLS constraints, as described in Section 4.

5.1 An Example of the Need to Coalesce

Relation SODL of Figure 8 contains two starship entities, each containing one tuple. After SODL is

Starship	KC	Objective	Destination	Last-Maintenance	TC
Enterprise	<i>S</i>	Diplomacy	Romulus	4-12-2003	<i>S</i>
Zardor	<i>S</i>	Warfare	Romulus	5-2-2003	<i>S</i>

Figure 8: SODL

updated with the insertion of the *Enterprise U*, it appears as in Figure 9.

Semantically, SODL now contains two MLS entities, the *Enterprise U* and the *Enterprise S*, which are polyinstantiations of each other. This semantic state may have been reached in one of (at least) two ways:

1. The *U* level inserted the *Enterprise U* because it **now believes a starship** named Enterprise exists which, in reality, is not the same starship as the *Enterprise S*. For example, a new starship may

Starship	KC	Objective	Destination	Last-Maintenance	TC
Enterprise	<i>U</i>	Exploration	null	4-12-2003	<i>U</i>
Enterprise	<i>S</i>	Diplomacy	Romulus	4-12-2003	<i>S</i>
Zardor	<i>S</i>	Warfare	Romulus	5-2-2003	<i>S</i>

Figure 9: *SODL* After Insertion

have been manufactured and inadvertently given the same name as the *Enterprise S* because the existence of the *Enterprise S* is unknown to the commissioner¹⁰.

2. The *U* level inserted the *Enterprise U* because information about the *Enterprise S* became available via a channel external to the database. For example, some information about the existence of a starship named Enterprise (the *Enterprise S*) was released to a newspaper and read by a *U* subject. In this case, both the *U* and *S* levels contain beliefs about the *same* starship, although they are not the same MLS entity in the database¹¹.

The *S* level must decide between these two alternatives. If *S* decides to believe the former case is actually true, nothing need be done to the database, since it accurately distinguishes between two distinct MLS entities, with distinct eids. If *S* decides to believe the latter case is true, the database should be altered to better reflect these beliefs. *S* must *coalesce* its beliefs with the *U* level beliefs, forming a single MLS entity.

5.2 COALESCE

In order to coalesce the *Enterprise S* with the *Enterprise U*, the *KC* attribute of the *Enterprise S* must be changed from *S* to *U*. Since the MLS-SQL UPDATE command does not permit changing an eid, we define the MLS-SQL COALESCE command as follows:

```
COALESCE      R
WHERE         P
BELIEVED BY  L;
```

Let COALESCE be issued by level *l*. The COALESCE statement is a variation of the UPDATE statement; WHERE and BELIEVED BY select a set of visible MLS entities, just as in UPDATE. For each selected entity, $KC < l$.

¹⁰ Called *entity polyinstantiation* in [6].

¹¹ Called *attribute polyinstantiation* in [6].

COALESCE has the following effect on the interpretation at level l . Consider an l -tuple t in the interpretation, bearing eid $K KC_i$. For each selected MLS entity with eid $K KC_j$ (where $KC_j < KC_i$), a new l -tuple t' is generated that is identical to t , except that KC_i is replaced with KC_j . The tuple t' is a *coalesced tuple*. This is performed for each l -tuple, until a set of newly generated coalesced tuples has been formed.

Property 1 could be violated by the addition of a tuple t' to the l interpretation if there already exists an l tuple t'' having eid $K KC_j$. If Property 1 would be violated, the coalesce is not performed and the newly generated tuples are discarded. Otherwise, the new tuples are added to the l interpretation, and each tuple in l from which coalesced tuples were generated is deleted¹². If no l tuple t exists with eid $K KC_i$, where a selected MLS entity has eid $K KC_j$ (where $KC_j < KC_i$), COALESCE has no effect on the l interpretation.

5.3 Using An MLS Rule To Coalesce

A number of factors may influence the decision to coalesce. For example, if S and U level subjects read the same newspaper stories, then the S subjects may infer that *Enterprise U* and *Enterprise S* are in fact the same entity. Inferences can also be made from the state of the database itself. For example, values in the *Enterprise U* tuple can lead to a decision: if the date of service is known to be fairly random, the identical last service date of the *Enterprise U* and *Enterprise S* can lead to the inference they are the same ship, and should be coalesced. This knowledge by the S level can be encoded in the following rule *Same-Starship*:

```

WHEN INSERTED SODL
IF      (SELECT * FROM SODL
        WHERE (Starship, Last-Maintenance) in
              (SELECT Starship, Last-Maintenance FROM INSERTED SODL ISODL
               WHERE ISODL.KC < SODL.KC
               BELIEVED BY ANYONE))
THEN COALESCE SODL
      WHERE (Starship, Last-Maintenance) in
            (SELECT Starship, Last-Maintenance FROM INSERTED SODL ISODL
             WHERE ISODL.KC < SODL.KC
             BELIEVED BY ANYONE)
      BELIEVED BY ANYONE;

```

¹²In practice, to maintain referential integrity, when entities are coalesced then the relationships these entities participate in also need updating.

Same-Starship monitors insertions (new MLS entities) in SODL. Whenever starships are inserted at a lower level than the rule activation, and their (apparent) key and Last-Maintenance date match a tuple at the rule's level, starships at the level of the rule are coalesced with those at the lower level¹³.

After the COALESCE of *Enterprise U* and *Enterprise S*, SODL appears as in Figure 10. The

Starship	KC	Objective	Destination	Last-Maintenance	TC
Enterprise	<i>U</i>	Exploration	null	4-12-2003	<i>U</i>
Enterprise	<i>U</i>	Diplomacy	Romulus	4-12-2003	<i>S</i>
Zardor	<i>S</i>	Warfare	Romulus	5-2-2003	<i>S</i>

Figure 10: *SODL* After Coalesce

first two tuples of this relation now form a single MLS entity for the *Enterprise U*.

Same-Starship at level *l* only coalesces the *l* level tuple of the *Enterprise S* with the *Enterprise U*. If *Enterprise S* were a multi-tuple MLS entity, *Same-Starship* would have to be activated at each level having an *Enterprise S* tuple to coalesce all tuples of the *Enterprise S* when *Last-Maintenance* dates match. However, each level decides for itself whether the *Same-Starship* policy is appropriate. If desired by another level, a rule performing a COALESCE using different criteria could be used, or no rule at all.

6 Conclusions

In this paper we have presented a model for *multilevel secure rules* modeled as MLS entities, in a single coherent security framework. We have shown how to extend an MLS relational database to incorporate active rules by introducing the special MLS Rules relation. Rules can be in the active or descriptive state, the latter permitting an exploratory (software engineering) approach to rule development. MLS rules respect the access restrictions of MLS security, and are designed to prevent covert storage and timing channels.

Despite restrictions imposed by multilevel security, a great deal of modeling power is available when rules are embedded in the MLS framework in the manner we have described. We have demonstrated, through rules like *Tracked-Ships* and *Watch-Neptune* of Section 2.4, and the rule *Same-Starship* used to illustrate a coalescing policy in Section 5.3, the use of multilevel secure rules in several different and useful situations. Rules can enhance the power and flexibility of a database without compromising the requirements of mandatory security.

¹³ *Same-Starship* employs a join between data at different levels of the database interpretation. Cross-level joins have been defined in [14].

References

- [1] Thomas A. Berson and Teresa F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings: 1987 IEEE Symposium on Security and Privacy*, pages 235–242, Oakland, CA, April 1987.
- [2] S. Ceri and Jennifer Widom. Deriving Production Rules for Constraint Maintenance. In *Proceedings: 16th VLDB Conference*, pages 566–577, Brisbane, Australia, August 1990.
- [3] Department of Defense, National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, 1985. DOD 5200.28-STD.
- [4] Thomas D. Garvey and Teresa F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings: 6th Computer Security Applications Conference*, pages 148–159, Tucson, December 1990.
- [5] L. Haas, W. Chang, G. Lohman, J. McPherson, P. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. Carey, and E. Shekita. Starburst Mid-flight: as the Dust Clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160, March 1990.
- [6] Teresa F. Lunt. Security in Database Systems: A Researcher's View. In *Proceedings: Second German Conference on Computer Security*, June 1991.
- [7] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, et al. The SeaView Security Model. *IEEE Transactions on Software Engineering*, 16(6):593–607, June 1990.
- [8] Teresa F. Lunt and Jonathan K. Millen. Secure Knowledge-Based Systems. Technical Report SRI-CSL-90-04, SRI International, August 1989.
- [9] Dennis R. McCarthy and Umeshwar Dayal. The Architecture of an Active Data Base Management System. In *Proceedings: ACM SIGMOD*, pages 215–224, Portland, Oregon, June 1989.
- [10] Matthew Morgenstern. Security and Inference in Multilevel Database and Knowledge-Base Systems. In *Proceedings: ACM SIGMOD*, pages 357–373, San Francisco, CA, May 1987.
- [11] Tore Risch. Monitoring Database Objects. In *Proceedings: 15th VLDB Conference*, pages 445–453, Amsterdam, Holland, 1989.
- [12] Ulf Schreier, Hamid Pirahesh, Rakesh Agrawal, and C. Mohan. Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS. In *Proceedings: 17th VLDB Conference*, Barcelona, Spain, 1991.
- [13] Ken Smith and Larry Jones. Efficient Propagation of Updates Using Join Numbers. Technical Report UIUCDCS-R-91-1663, University of Illinois at Urbana-Champaign, March 1991.

- [14] Ken Smith and Marianne Winslett. A Multilevel Secure Relational Data Model. (In Preparation for TODS), 1992.
- [15] Ken Smith and Marianne Winslett. Entity Modeling in the MLS Relational Model. In *Proceedings: 18th VLDB*, page ??, Vancouver, British Columbia, Canada, August 1992.
- [16] Michael Stonebraker, Eric N. Hanson, and Spyros Potamianos. The POSTGRES Rule Manager. *IEEE Transactions on Software Engineering*, 14(7):897–907, July 1988.
- [17] Jennifer Widom, R. Cochrane, and Bruce Lindsay. Implementing Set-Oriented Production Rules as an Extension to Starburst. In *Proceedings: 17th VLDB*, pages 275–285, Barcelona, Spain, September 1991.
- [18] Jennifer Widom and Sheldon J. Finkelstein. A Syntax and Semantics for Set-Oriented Production Rules in Relational Database Systems. In *Proceedings: ACM SIGMOD*, pages 259–270, Atlantic City, N. J., May 1990.
- [19] Jackson Wilson. A Security Policy for an A1 DBMS (a Trusted Subject). In *Proceedings: IEEE Symposium on Research in Security and Privacy*, pages 116–125, Oakland, CA, May 1989.

The SPEAR Data Design Method

Peter J. Sell

Office of INFOSEC Computer Science, Department of Defense, 9800 Savage Road,
Fort George G. Meade, Maryland 20755, USA

Abstract

This paper presents the SPEAR Data Design Method of specifying multilevel database applications based on the SPEAR Data Model. This data design method allows a database designer to decompose an application into entities and relationships and graphically represent the application using a SPEAR Data Diagram. An example multilevel database application is given to illustrate the method. In addition, the SPEAR Data Design Method is compared to the Semantic Data Model for Security (SDMS), another multilevel specification notation.

1.0 INTRODUCTION

Designing a database application is a complex process that becomes more difficult when the application is being designed for a multilevel secure database management system (DBMS). In order to simplify this difficult task, it is necessary to look at the application at an abstract level. By doing so, the database application designer gains a greater understanding of the application and, therefore, lessens the chance of an insecure design or implementation of the application.

Most existing data modelling methods[Chen76] are not designed to handle multilevel data. The result of using an inappropriate method to design multilevel applications can lead to mistakes. Past attempts to create new data design methods to capture security features [Smith90] have not been adequate to specify all applications.

The SPEAR Abstract Data Model [Wisem91] provides a model capable of describing applications at an abstract level. The SPEAR model does not provide a notation for the specification of applications. The SPEAR Data Design Method, therefore, attempts to provide such a notation for specifying database application based on the SPEAR Abstract Data Model as well as, the Semantic Data Model for Security (SDMS), a previous data modeling approach [Smith90].

The SPEAR Data Design Method allows a designer to model an application in terms of its entities and relationships and produce a SPEAR data diagram, a graphical specification of the application. This diagram includes classification information, entity relationships, and attribute constraints.

This paper provides an overview of the SPEAR Abstract Data Model, describes the SPEAR Data Design Method, compares the SPEAR Data Design Method with SDMS, and finally provides an example of an application specified using the SPEAR Data Design Method.

2.0 THE SPEAR ABSTRACT DATA MODEL

Simon Wiseman of the Defence Research Agency (DRA) in Malvern, England developed a data model that allows database applications to be specified in terms of entities, classes, relationships, families and parties. This model is known as the SPEAR Abstract Data Model and is based on the E-R model developed by Peter Pin-Shan Chen [Chen76].

2.1 Entities and relationships

The SPEAR data model uses abstraction to decompose the application into entities and relationships. An entity is a "real world" object that can be described in terms of its attributes. For example an entity may be a "car" or a "person". Some of the attributes of a car include a vehicle identification number, a model, and color. The attributes of a person may include a name, a birthdate, and shoe size.

The value of an attribute for an entity is a set of values and is referred to as the attribute's value set. This feature allows the values for an attribute to be handled as a single object. For example, a car may have multiple colors such as a two-tone paint combination. The number of members in an attribute's value set can be limited to one. This corresponds to the standard E-R model [Chen76].

A relationship is between two or more entities. For example, a relationship between a "person" and a "car" may be "drives". In this case, the "drives" relationship is capable of identifying the person driving a car and the car driven by a person.

2.2 Classes and families

Database applications are typically concerned with groups of related entities. The SPEAR model, therefore, groups entities together into classes of entities. The entities in a class are described by assigning values to their attributes. An example of a class of entities may be "People". Each person in the class will have the same attributes (i.e. name, birthdate, and shoe_size), but the values for these attributes may be different. In some cases two entities may have the same values assigned to their attributes, e.g. two Joan Smith's, born on 1 January 1960 and have a size 7 shoe may exist in the same class of people.

A relationship is formed within two or more entities. This method, therefore, groups relationships together into families of relationships. In this case, a family will include all of the individual relationships between two or more classes. An example may be a family named "drives" that relates the class of "cars" and the class of "people". The "drives" family is able to identify the individuals that drive cars and the cars that are driven by people.

2.3 Levels of classification

The SPEAR Data Model allows the designer to specify classification constraints at the following levels:

- 1) The existence of a class of entities or family of relationships.
- 2) The name of a class or family.
- 3) The number of individual entities in a class, (for example, knowing that 10 cars are included in the entity named "cars") or individual relationships in a family (knowing that 5 people are driving cars).
- 4) The name and existence of an attribute and the name of the domain of the attribute.
- 5) The existence of a value for a particular attribute.
- 6) The actual value of a particular attribute.

Each of these six levels of classification has a corresponding classification in the SPEAR Data Design Method. This mapping is discussed in the next section.

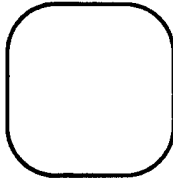
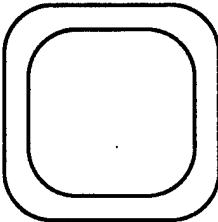
Type of Class	Graphical Notation
Dynamic Class	
Static Class	

Figure 1. Graphical Notation for Classes.

3.0 THE SPEAR DATA DESIGN METHOD

The SPEAR Data Design Method uses abstraction to simplify the application by ignoring unnecessary details while focusing on the structure of the data and its classification. Specifying the application using this design method consists of four steps:

- 1) Identify the classes, families, and attributes of the application,
- 2) Assign classification levels to the classes, families, and attributes,
- 3) Identify the constraints on the attributes, and
- 4) Construct a SPEAR diagram to graphically represent the application.

This notation allows the database designer to graphically display the application, including all classification and attribute constraints. The graphical representation of an application is known as a SPEAR Data Diagram and such diagrams contains four major graphical notations: classes, families, inheritance, and connections.

3.1 Class and family graphical notation

Figure 1 shows the two graphical symbols used to denote the types of classes. The single outline of the box denotes that it is a dynamic class (i.e. the values of the attributes may change and entities may be added and deleted). An example of a dynamic class may be the cars built in 1982. In this example, values for the attributes of the car may change (i.e. the car may be repainted in a different color) or cars may be deleted from the class as they are sent to the junk yard. A static class (i.e. the values of its attributes do not change and entities cannot be added or deleted) is represented by a double outline. An example of a static class may be the models of cars manufactured in 1982. In this case, the values of the models of cars do not change, since this is historical information.

Figure 2 shows the graphical notations for family diagrams. The single outline of the box denotes that its a dynamic family (i.e. the values of the attributes are allowed to change

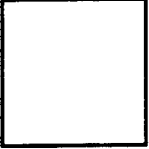
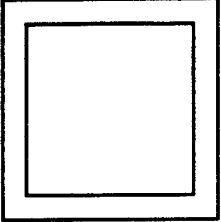
Type of Family	Graphical Notation
Dynamic Family	
Static Family	

Figure 2. Graphical Notation for Families.

and relationships can be added or deleted). A static family (i.e. the values of its attributes do not change and relationships cannot be added or deleted) is represented by a double outline.

Both the class diagram and family diagram have three distinct parts (see Figure 3): the header section, the attribute section, and the constraint section.

3.1.1 The header section

The elements associated with the header section of the class or family are defined as follows:

Class_name is the name of the class and **Family_name** is the name of the family. The names of classes and families must be unique within the application.

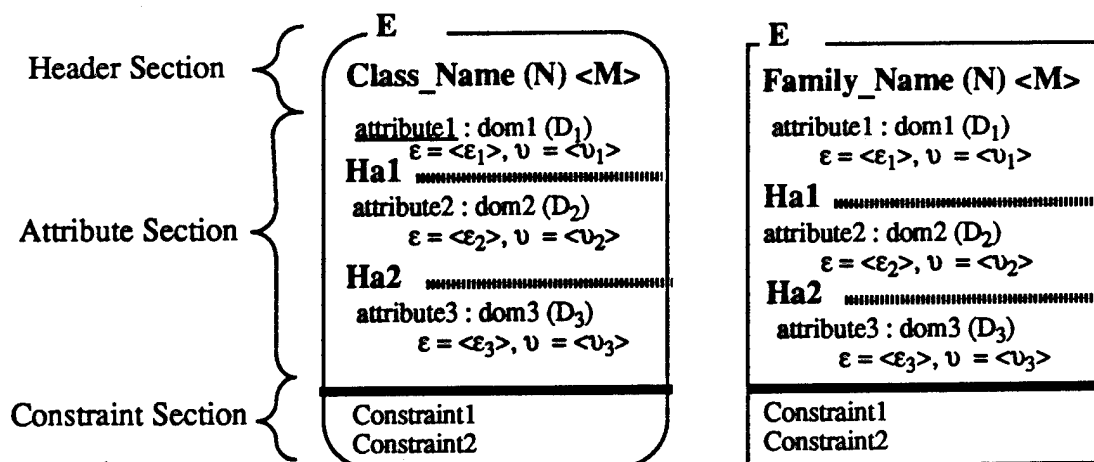


Figure 3. Class and Family Diagrams.

E is the classification of the existence of the class or family.

N is the classification of the name of the class or family. It also is the default classification of the existence of the attributes of the class or family. In this case only, the classification of **Attribute1**'s existence is **N**. (See below for classifying the existence of other attributes.) Note that the classification of the existence of the class or family (**E**) must be dominated by (\leq) the classification of the name (**N**), i.e. $N \geq E$. If **N** is not specified, the value of **N** defaults to the classification of the existence of the class or family (**E**).

M is the set of classifications that classify the existence of an individual member of the class or family. It follows that to know the existence of an individual entity or relationship, it is necessary to know the existence of the class or family, therefore, $M \geq E$. If **M** is not specified, the value of **M** defaults to the classification of the name of the class or family (**N**)

3.1.2 The attribute section

The elements associated with the attribute section of the class or family are defined as follows:

attribute1 is the name of the first attribute that describes the entities of **Class_name** and the relationships of **Family_name**. The underlined attribute name indicates that a single uniqueness constraint exists on the value sets of that attribute, i.e. this is the primary key for the class and family. That is, no two entities can have the same set of values for **attribute1**. A composite uniqueness constraint is declared by underlining multiple attribute names. Uniqueness constraints may also be declared by placing the attribute name in the constraint portion of the box. Uniqueness constraints are optional.

attribute2 and **attribute3** are names of additional attributes that describe the class and family. The names of attributes must be unique within a class or family, but need not be unique within the application. In this example, the value sets of **attribute2** and **attribute3** are not unique.

dom1, **dom2**, and **dom3** are the names of the domains for the corresponding attribute sets.

Ha₁ is the classification for the existence of the hidden attributes beneath the first dashed line. In this case, a user with a clearance that is not dominated by **Ha₁** is not able to know that "**attribute2**" exists.

Ha₂ is the classification for the existence of the attributes beneath the next dashed line. In this case, a user with a clearance that is not dominated by **Ha₂** is not able to detect that "**attribute3**" exists.

There is no limit to the number of dashed lines that may be placed inside a class or family or the number of attributes below the line.

D₁ is the classification of knowing both that **attribute1** is the name of an attribute which has a domain of "**dom1**" and that it is the primary key attribute of the class "**Class_name**" and the primary key attribute for the family "**Family_name**". If **D₁** is not specified, the value of **D₁** defaults to the classification of the name of the class or family (**N**)

D₂ and **D₃** is the classifications of knowing the attribute and domain names for the second and third attributes of "**Class_name**" and "**Family_name**". If **D₂** or **D₃** is not specified, the value of **N** defaults to the classification of the existence of that hidden attribute (**Ha₁** for **D₂** and **Ha₂** for **D₃**)

ϵ_n is the set of classifications permissible for the existence of a particular value of an attribute. If **ϵ_n** is not specified, the value of **ϵ_n** defaults to the classification of the corresponding attribute's name (**D_n**)

v_n is the set of classifications that a value of an attribute is allowed to have. If **v_n** is not specified, the value of **v_n** defaults to the classification of the existence of values for that attribute (**ϵ_n**)

Information associated with each attribute can be labelled to permit the following user accesses:

- a) The user does not know that an attribute exists, i.e. inspection of an entity may reveal one or more attributes, but the user is unaware of the existence of further attributes.
- b) The user may know that an attribute exists, but have no knowledge about its name or domain.
- c) The user knows of the existence, name, and domain of the attribute.
- d) The user knows of the existence, name, and domain of an attribute and is able to detect the existence of some of the values for the attribute.
- e) The user knows of the existence, name, and domain of an attribute and is able to see some of the values for the attribute.

From a-e it follows that:

Classification of a value in an attribute's value set \geq

Classification of the existence of a value in an attribute's value set \geq

Classification of an attribute's name and domain name \geq

Classification of an attribute's existence \geq

Classification of the existence of the class or family

Each of the above cases can be achieved by appropriately setting $D_1 \dots D_n$, E , $Ha_1 \dots Ha_n$, $\epsilon_1 \dots \epsilon_n$, and $v_1 \dots v_n$.

Knowledge of information protected by D_1 includes information protected by N . Therefore, the user must be permitted to know about "Class_name" or "Family_name" before being given access to the information protected by D_1 , i.e. $D_1 \geq N$.

Consequently, $D_2 \geq Ha_1$.

Since E is protecting the existence of some of the attributes, it therefore follows that D_i must dominate E , i.e. $D_i \geq E$.

Ha_1 protects the existence of an attribute below the dashed line. Therefore $D_2 \geq Ha_1$, since knowing the information protected by D_2 includes knowledge of the existence of the attribute. As the existence of "attribute2" is to be hidden from users able to see the proceeding attributes, it follows that $Ha_1 \geq E$.

Ha_2 protects the existence of an attribute below the second dashed line. Therefore $D_3 \geq Ha_2$, since knowing the information protected by D_3 includes knowledge of the existence of the attribute. As the existence of "attribute3" is to be hidden from users able to see the preceding attributes, it follows that either $Ha_2 \geq Ha_1$ or Ha_1 and Ha_2 are not comparable. If they are not comparable, problems may appear during implementation.

ϵ_n protects the existence of a particular value of an attribute. Therefore, it follows that for an individual element of an entity's or relationship's attribute value set $\epsilon \geq D_n$.

v_n protects the values of an attribute. Since it is necessary to know the existence of a value before knowing the actual value, it follows that for an individual element of an entity's or relationship's attribute value set, $v \geq \epsilon_n$ for a particular n .

The following table summarizes the classification constraints:

$D_n \geq Ha_m \geq E$, where Ha_m is the classification of the dashed line corresponding to D_n

$D_1 \geq N \geq E$

$v \geq \epsilon \geq D_n$, for an individual element of an entity's or relationship's value set.

$Ha_n \geq Ha_{n-1}$

3.1.3 The constraint section

Below the solid line, constraints on the attributes are stated. Constraints can be one of four forms:

1) attribute_name > value indicates the value of attribute_name must be greater than value.

2) **low < attribute_name < high** indicates the value of attribute_name is between low and high.

3) **#attribute_name >= 1**, indicates that the attribute is allowed to have 1 or more values in its attribute set. By default, the number of individual elements in an attribute's value set is one.

4) **attribute_name**, indicates that the values in the attribute's value set must be unique. Several attribute names may be underlined to indicate a multiple uniqueness constraint.

In order to reference an attribute of another class or family, it is necessary to include its unique class or family name, since attribute names are not unique within the entire application. This is accomplished by preceding the attribute name with the name of its class or family.

In the constraints section, a long reference to an attribute can be shortened by using a macro definition. A macro definition contains the attribute name, enclosed in quotes (") followed by \equiv and the macro name. A reference to a person's age in a class named people can be shortened by the following macro definition:

"People.shoe_size" \equiv size

3.2 Examples of classes

Figure 4 shows two classes, "Cars" and "People". Default classifications have not been used in this diagram. Figure 5 illustrates the use of default classifications. Each car in the class of "Cars" can be described using three attributes, "Vehicle_ID", "Model", and "Color". In this class, all of the vehicle identifiers must belong to the domain of integers, the model is a character string, and the color is in the domain of colors. In addition, the values of "Vehicle_ID" must be unique. All the information is unclassified. The "Color" attribute has an integrity constraint that the number of individual colors in a value set must be at least one. This allows a car to have multiple colors, e.g. a two-tone paint combination.

Persons in the "People" class are described using three attributes: "Name", "Birthday", and "Shoe_size". All of this information is unclassified, except the actual values of "Shoe_size" are confidential. In the "People" class the "Shoe_size" attribute's value must be greater than zero.

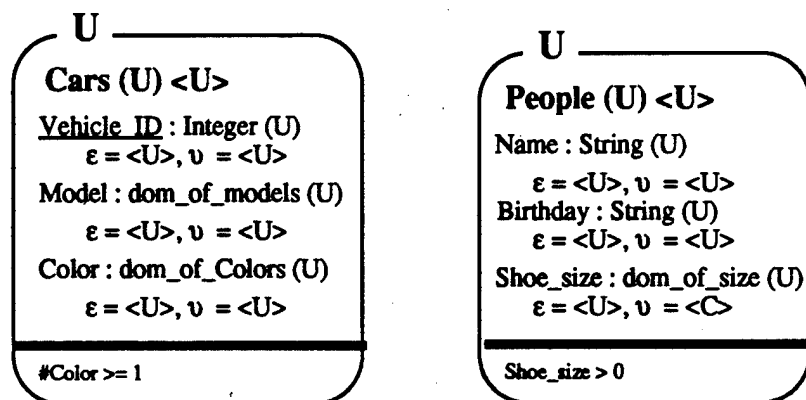


Figure 4. Examples of a Class of Cars and a Class of People

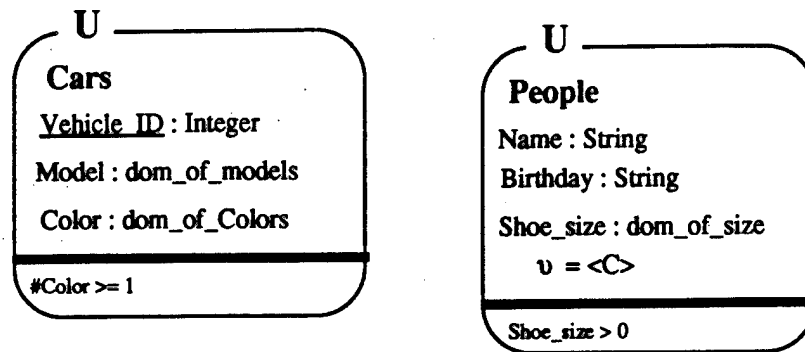


Figure 5. Examples of a Class of Cars and a Class of People Using Default Classifications.

3.3 Example of a family

Figure 6 Shows an example of a "Drives" family. This family does not contain any attributes or constraints. All information about the family is unclassified.

3.4 Inheritance

Two or more classes can be associated with an inheritance. Inheritance indicates that the classes are involved in a hierarchy, with the child class inheriting all the attributes and constraints of the parent class. If an entity is inherited by a subclass, that entity must have all of the attributes and constraints of the parent class. This type of inheritance is represented as a hollow arrow pointing from a subclass to its parent class. Figure 7 shows the graphical notation for types of inheritance. For example, a parent class contains cars. A sub class may be either four wheeled cars or three wheeled cars. An entity in the four wheeled car class will inherit all of the attributes of the car class, such as engine, steering wheel, brakes, and seats.

The arrow may have a dashed or solid outline. The dashed outline indicates that an entity in the parent class does not have to be a member of the subclass. A solid outline indicates that each of the entities in the parent class must be an entity in the subclass.

The arrow may have an arrow head on both ends. This indicates that the inheritance is two way. The parent class inherits all of the attributes and constraints of the sub class and the subclass inherits all of the attributes and constraints of the parent class, i.e. both classes have the same attributes and constraints.

An arc may intersect the inheritance arrows, this indicates that an individual entity from the parent class may belong to at most one subclass associated with the arc. An association with out an arc indicates that an individual entity from the parent class may belong to more than one sub class.

3.5 Connections

A class is connected to a family by a dashed or solid line. A solid line indicates that the entire set of entities of the class must be a party to some relationship in the family. A

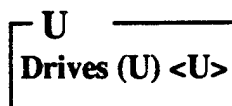


Figure 6. The Drives Family

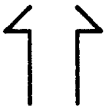

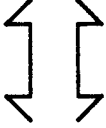

Type of Inheritance	Graphical Notation
Mandatory One-Way	
Optional One-Way	
Mandatory Two-Way	
Optional Two-Way	

Figure 7. Inheritance Notation.

dashed line denotes that only a partial set of entities of the connected class participates in the family. Figure 8 shows the graphical notation for the types of family participation.

Attached to this line is the name of the party involved in the relationships. The name of the party may be underlined indicating that all the sets of values from the participating class are unique.

Following the name of the party can be a number or a set of numbers in parentheses. These numbers indicate the number of individual elements in the party set allowed to participate in the family. A name without a number following it indicates that only one element may participate. It is also possible to require that the number of elements be within some range. The notation for this is the low number followed by '..' followed by the high number (e.g. 1..10).

An arc through one or more lines indicates that the individual objects participating in the family are mutually exclusive. In other words, all the individual elements of all sets of



Type of Participation	Graphical Notation
Mandatory Participation	
Optional Participation	

Figure 8. Connections Between Entities and Families

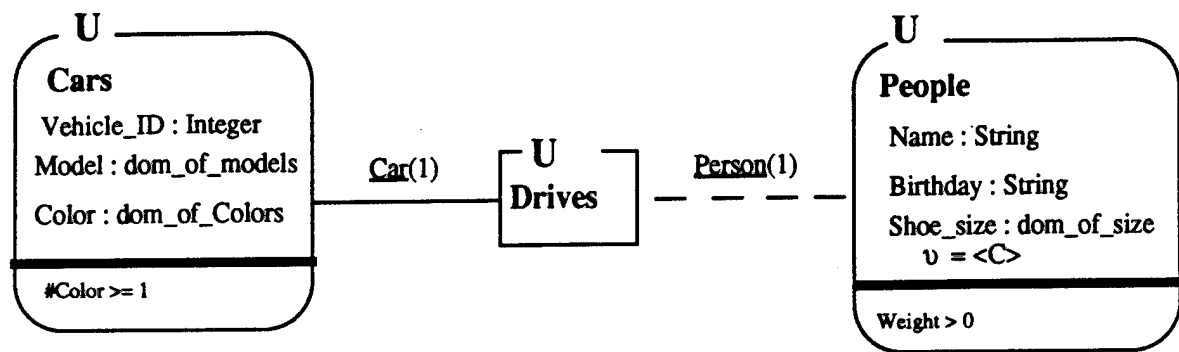


Figure 9. SPEAR Diagram Example

parties are unique. For example, an individual car cannot be driven by more than one person at time. Similarly, a person can not drive more than one car at a time.

Figure 9 shows an example of a possible connection between the "People" class and the "Drives" family and the "Cars" class. The solid line between the "Cars" class and the "Drives" family indicates that every car must participate in the family. The party from the "Cars" class is a "Car" and the party from the "People" class is a "person". The family relates all of the cars to some of the people. An individual relationship relates a single car and a single person. The dashed line between the "Drives" family and the "People" class indicates that not all people must drive a car.

Figure 10 shows an abbreviated SPEAR diagram. This notation consists of the basic class and family boxes and the connections between them. All classifications and attribute information is deleted. This notation is necessary when showing the overall application where details may be omitted for clarity.

4.0 SEMANTIC DATA MODEL FOR SECURITY (SDMS) COMPARISON

Gary Smith of the National Defense University developed the Semantic Data Model for Security (SDMS) [Smith90].

The following areas of SDMS have been enhanced in the SPEAR notation:

- 1) Arrowheads
- 2) Attributes inside the entity notation
- 3) Thickness of lines
- 4) Expanded classification levels

Figure 10 describes the "People" and "Cars" classes described in Figure 8 using the SDMS notation. This example is used as a comparison between the SPEAR notation and the SDMS notation.



Figure 10. Example of an Abbreviated SPEAR Diagram

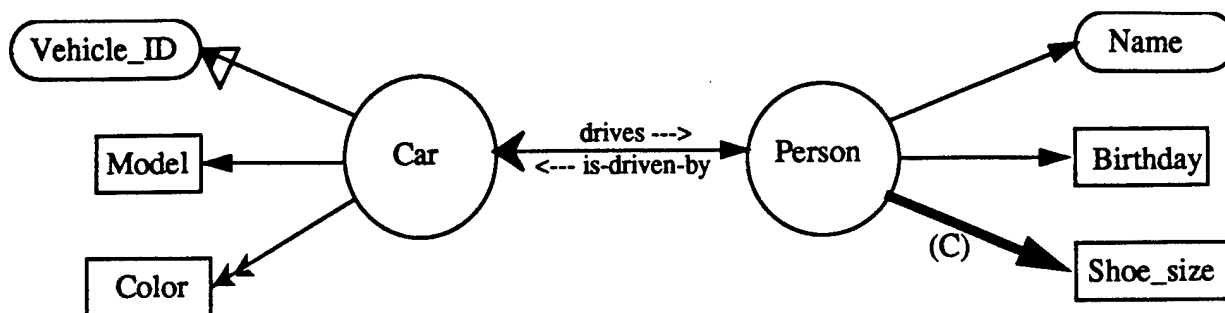


Figure 11. Example of the SDMS Notation

4.1 Arrowheads

The SDMS model uses five different types of arrowheads to graphically represent integrity property constraints on the relationships. Figure 11 shows four of these types. As can be seen in Figure 8, the SPEAR notation does not rely on arrowheads to provide information. This has the benefit of simplifying the SPEAR diagram.

4.2 Placement of attributes

The SDMS notation physically separates the attributes from the entity. The SPEAR notation has included the attributes within the class construct.

4.3 Line boldness

The SDMS notation has added classification information to the thickness of an association line. In this notation classified associations are indicated with a bold line. An example of this can be found in Figure 11. The arrow that associates "Person" with "Shoe_size" is bold and has a confidential classification attached to this line.

In the SPEAR notation, classification information is always included within the family or class notation. The thickness of lines in the SPEAR notation are irrelevant. One reason for this decision is that when photocopying the diagram the thickness of lines become less distinct, and therefore, classification information may be lost.

4.4 Classification levels

The SDMS model does not allow the separation of the existence of a value for an attribute and the actual value of the attribute. This requirement is part of the SPEAR model and SPEAR notation.

5.0 Sample application

The use of the SPEAR Data Design Method can be illustrated through a sample application involving the movement of cargo and personnel by ship.

5.1 Description of application

This application is a simplified version of the application described in "Crisis Management Sample Application for SWORD" [SellLewis91a]. This shortened version of the crisis management application involves the movement of cargo and personnel by ship. Orders are placed on ships and ships move between ports. An order can be one of four types: 1) manoeuver order, 2) repair order, 3) load or unload order, or 4) wait order.

The manoeuver order requests that the ship report to a specific port. The repair order specifies that the ship report to a specific port for the specified repairs. The load or unload order requests that the ship either load personnel or cargo indicated by a positive number or unload personnel or cargo indicated by a negative number. The wait order requires the ship to remain idle at its current port until that ship is notified of the completion of an event. For example, five ships may be assigned to leave port together, but one ship is in the process of unloading cargo. All five ships will be issued a wait order. Four of these ships will be waiting, while the fifth ship is completing its current order. After the ship completes its current unload order and executes its next order, the wait order, all five ships will now be able to execute their next order, in this case a maneuver order. A ship may be either in a port, being repaired at a port or sailing to a port at any one time. A ship may be idle, not executing an order, or have many orders waiting to be executed.

Four different types of users will have access to this application: captains of ships, naval chiefs, maintenance coordinators, and journalists.

5.1.1 Details of orders

All orders contain a unique order number, the name of the ship the order is assigned to, and a priority number (the higher the number, the lower the priority). A manoeuver order also contains a destination. A repair order also contains a destination, a description of repairs, a time interval, and an actual description of the repairs. A load or unload order also contains change in equipment, change in personnel, and a time interval. A wait order also contains the name of the event.

The following are examples of orders:

Ship_A report to Port_2 , Priority=1

Ship_C load 20 personnel and 30 tons of equipment, time interval = 7 hours,
Priority = 2

Ship_D load -30 personnel and -10 tons of equipment, time interval = 3 hours,
Priority = 3

Ship_B report to Port_3 for repairs (Engine Repairs), time interval = 5 hours,
Priority = 4

Ship_E wait for Event_2, Priority = 1

The knowledge that the orders exist and that they are orders on the fleet are unclassified. In addition, the knowledge of the total number of orders in the application is also unclassified. The name and existence of values for the order number, the assigned ship, and the priority number are all unclassified. The actual values for the order number and assigned ship are unclassified. The actual values for the priority number is confidential.

The existence of manoeuver orders, repair orders and load or unload orders are unclassified. The existence of a wait order is considered confidential.

For a manoeuver order the name, ability to count the manoeuver orders, knowing that a destination is an attribute and the existence of a destination value are all unclassified information. The actual value for a destination is secret.

For a repair order the name and ability to count the total number of repair orders are unclassified. Knowing that a repair order contains a destination, description or repairs and a time interval is unclassified, but the knowledge of the existence of an actual description is confidential information. the existence of values for the destination, description, and time interval are all unclassified. The actual values for the destination and description are both unclassified. The values of the time interval are confidential. The name of the actual description attribute and existence of values are confidential. Actual values for the actual description can be either confidential or secret. The description in this case is actually a cover story [Nelso91] [Wisem91a] for the actual repair description.

For a load or unload order the name, ability to count the load or unload orders, knowing that change in equipment, change in personnel, and time interval are attributes and the existence of values for the three attributes are all unclassified information. The actual

values for change in equipment and change in personnel are unclassified. The actual values for time interval are confidential.

For a wait order, the name and ability to count the total number of wait orders are confidential. Knowing that a wait order contains an event name and the existence of a value for the event name is confidential. The actual event names may be either confidential or secret.

5.1.2 Details of ship types

All ships belong to a category of ship (i.e. destroyer, aircraft carrier,...). Knowing that types of ships exists is unclassified. Each type of ship includes a name, a maximum capacity for equipment and personnel and a maximum speed. The knowledge of the name "ship_type" is unclassified as is the ability to determine the total number of types of ships. Knowing that a type of ships has a name, maximum equipment capacity, maximum personnel capacity, and a maximum speed are all unclassified. The existence of values for all four of these attributes is also unclassified. The actual values for name, maximum equipment, and maximum personnel are unclassified. The value for maximum speed is confidential.

5.1.3 Details of ships

Each ship has a unique name and a cargo of equipment and personnel. The name "ship" and the ability to count the number of ships are unclassified. Knowing that a ship has a name, equipment, and personnel is unclassified. the existence of values for the name, equipment, and personnel are all unclassified. Actual names of ships are unclassified and actual values for equipment and personnel are confidential.

A ship can be either in port, underway at sea, or being repaired. Knowing that a ship can be in one of these locations is unclassified. For a ship in port, the name and ability to count the number of ships in port is unclassified. A ship in port has two unclassified attributes, expected completion time and current port. The existence of values for these attributes are unclassified. The actual value for the expected completion time is either unclassified or confidential. The actual value for the current port is confidential.

For a ship underway, the name and ability to count the ships underway is unclassified. A ship underway has an unclassified attribute for it, estimated time of arrival (eta). The existence of an eta value is unclassified and its actual value is confidential.

Knowing that ships can be repaired and knowing the number of ships being repaired is unclassified. Ships being repaired have two unclassified attributes, port they are currently at and expected completion time. The existence of values for the two attributes is unclassified and the actual values are confidential.

5.1.4 Port details

Knowing that ports exist is unclassified. Knowing the number of ports is also unclassified. Each port has a name which is unclassified.

5.2 The SPEAR diagram

The first step in producing a SPEAR diagram is to identify the classes of entities and the families of relationships. The class of entities and their attributes are:

- 1) Orders,
 - a) Order number
 - b) To ship
 - c) Priority
- 2) Manoeuver Orders (sub-class of Orders)
 - a) Destination
- 3) Repair Orders (sub-class of Orders)
 - a) Destination
 - b) Description
 - c) Time interval

- d) Actual description (Hidden Attribute)
- 4) Load and Unload Orders (sub-class of Orders)
 - a) Change in equipment
 - b) Change in personnel
 - c) Time interval
- 5) Wait Orders (sub-class of Orders)
 - a) Event name
- 6) Ships
 - a) Name
 - b) Current equipment
 - c) Current Personnel
- 7) Ships in Port (sub-class of Ships)
 - a) Expected completion
 - b) Current port
- 8) Ships Underway (sub-class of Ships)
 - a) Eta
- 9) Ships Being Repaired (sub-class of Ships)
 - a) Port
 - b) Expected completion
- 10) Ports
 - a) Port name
- 11) Ship Types (Static class)
 - a) Type name
 - b) Maximum equipment
 - c) Maximum personnel
 - d) Maximum speed
- 12) Order Status
- 13) Executing Orders
- 14) Queued Orders

The families in this application are:

- 1) Executes (between Executing Orders and Ships)
- 2) Order queue (between Queued Orders and Ships)
- 3) Is_a (between Ships and Ship Types)

Figure 12. shows the overall SPEAR Data Diagram for the application.

Steps two, three and four have been combined to produce the SPEAR Diagram in Figures 13 and 14. Although the diagram has been separated over two pages, the overall diagram of the application found in Figure 12 can be used to connect the two diagrams. In this case, the diagram are connected using the Order_Status class in Figure 13 and the Order class in Figure 14.

6.0 Summary

The purpose of the SPEAR Data Design Method is to provide a means of specifying multilevel database applications using the SPEAR Data Model. The SPEAR model allows six levels of classifications, that allows a database designer the flexibility to fine tune the classification levels of the data to fit operational requirements.

The SPEAR Data Design Method allows the database designer to analyze the application in abstract terms. The application is first decomposed into classes of entities and

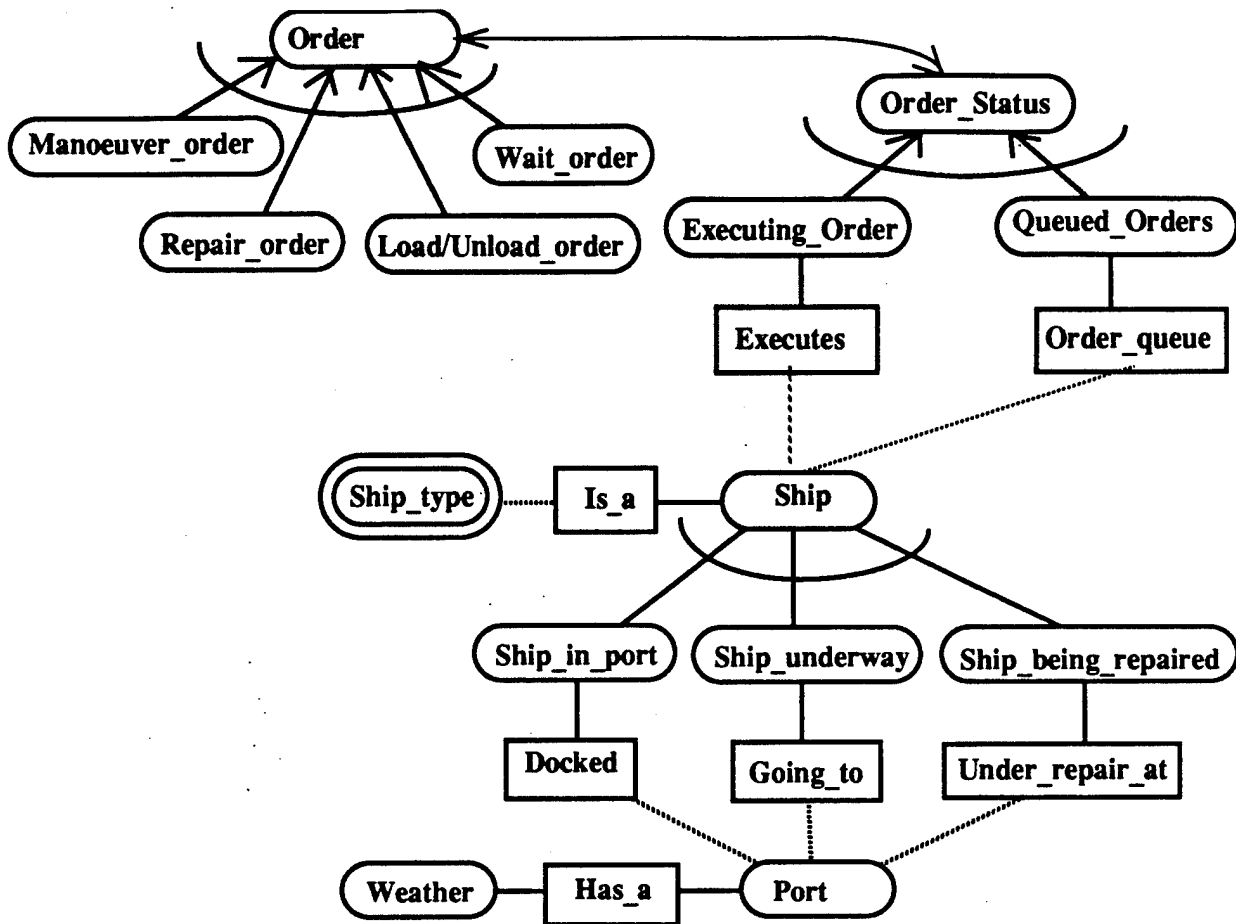


Figure 12. Abbreviated SPEAR Diagram for Sample Application

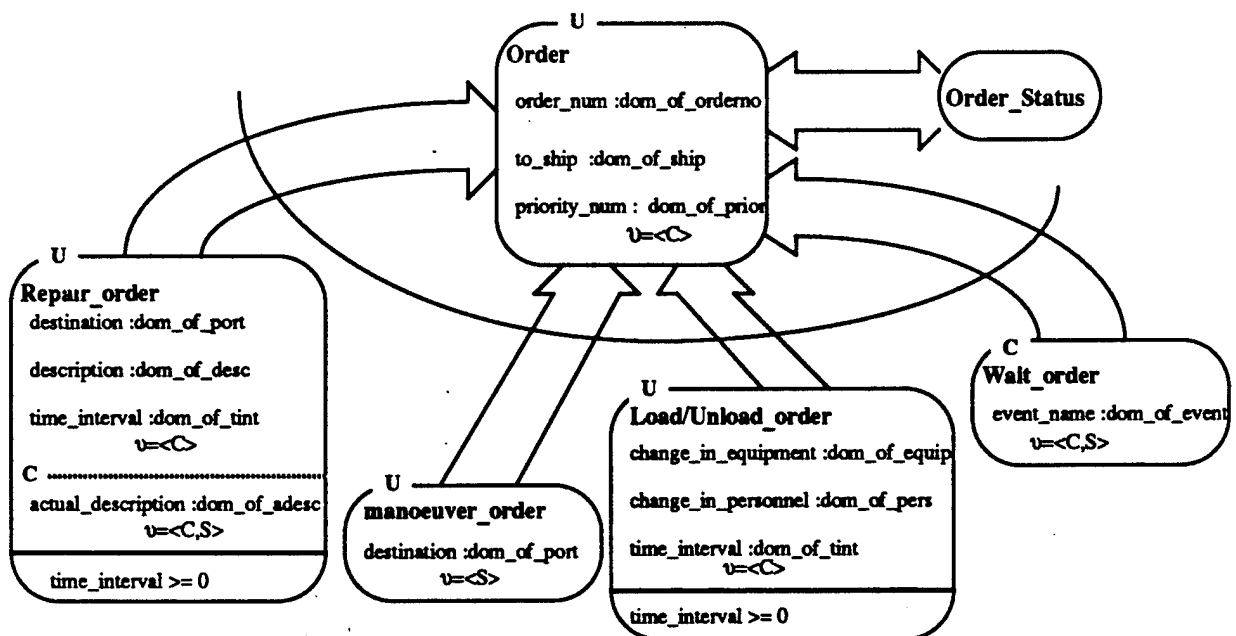


Figure 13. Order Section Of Diagram



families of relationships. The attributes of the entities and constraints on those attributes are then determined. Once the constraints on the attributes have been specified, a SPEAR Diagram graphically specifying the application is produced.

The SPEAR Data Design Method provides several enhancements to the SDMS notation. Among these enhancements are the reduction in the number of types of arrowheads, elimination of difference in line thicknesses, including an entity's attributes within the class notation, and expanded levels of classifications.

7.0 Acknowledgments

The SPEAR Data Design Method was developed under a scientist exchange program between the United States Department of Defense and the Defence Research Agency (DRA) in Malvern, England.

8.0 References

- [Chen76] Chen, Peter Pin-Shan, "The Entity-Relationship Model -- Toward a Unified View of Data," ACM Transactions on Database Systems, Vol. 1 No. 1, March 1976, pp. 9-36.
- [Nelso91] Nelson, D. and C. Paradise, "Using Polyinstantiation to develop an MLS Application," Proceedings of the 7th Annual Computer Security Applications Conference, San Antonio, Texas, December 1991, pp. 12-22.
- [SellLewis91a] Sell, Peter J. and Sharon Lewis, "Crisis Management Sample Application for SWORD," Draft, 21 November 1991.
- [SellLewis91b] Sell, Peter J. and Sharon Lewis, "The Notation for the SPEAR Data Design Method," Draft, 21 November 1991.
- [Smith90] Smith, Gary W., "The Modeling and Representation of Security Semantics for Database Applications," Thesis, George Mason University, Fairfax, Virginia, Spring 1990.
- [Wisem91a] Wiseman, Simon "Lies, Damned Lies and Databases," RSRE Memorandum 4503, July 1991.
- [Wisem91] Wiseman, Simon, "Abstract and Concrete Models for Secure Database Applications," Proceedings of the Fifth IFIP WG11.3 Working Conference on Database Security, Shepherdstown, West Virginia, November 1991.

Using SWORD for the Military Airlift Command example database

Simon R. Wiseman

Defence Research Agency, Malvern, Worcestershire WR14 3PS, England

Copyright © British Crown Copyright 1992

Abstract

It is widely thought that secure applications requiring cover stories must use a DBMS that forces the application to polyinstantiate. An example of the use of cover stories is given and it is shown that this can be implemented satisfactorily, without resorting to polyinstantiation, by using the SWORD secure DBMS. The example application is modelled abstractly, using a form of the Entity-Relationship model extended to cover security issues. The SWORD implementation is described and includes view definitions that ease its use by applications, and trigger definitions which add general purpose integrity constraints.

1. INTRODUCTION

In this paper the Military Airlift Command Example (MACE) is described and it is shown how the requirements can be met using the SWORD secure relational DBMS [Wood92]. The requirement for cover stories is a particular feature of MACE and the goal of this paper is to show how cover stories can be provided using a secure DBMS that does not require applications to polyinstantiate [Wiseman91b].

MACE is a simple system for maintaining logistics information about aircraft. It is based on the requirements of the United States Transportation Command / Military Airlift Command (USTRANSCOM/MAC), as described in [Nelson91]. The reason for using a fictional example, rather than the real thing, is that the USTRANSCOM/MAC requirements are not publicly available.

A prototype implementation of USTRANSCOM/MAC's requirements has been implemented using the Sybase Secure DBMS hosted on VAX/SE-VMS

[Nelson91]. This prototyping work was deliberately based on a DBMS that requires the application to polyinstantiate. This decision may well have been prompted by the wish to use a commercial, off-the-shelf secure DBMS. Since the majority of these are incapable of supporting general uniqueness constraints¹, the application will be forced to polyinstantiate. In addition, the requirements for the system include the need to provide cover stores, and it is widely believed throughout the research community that polyinstantiation is a technique that has been designed to support cover stories [Smith89§2.1] [Jajodia90§4.2], and even that it is essential for supporting them.

The SWORD secure DBMS, which currently exists only as a research prototype [Lewis91], does support general uniqueness constraints and so an application does not have to resort to polyinstantiation. However, SWORD appears perfectly capable of supporting the USTRANSCOM/MAC requirements, including those for cover stories. Indeed, it is argued in [Wiseman91a] that polyinstantiation is a poor technique for cover stories, since it is difficult to prevent them arising spuriously.

This paper describes the MACE application in section 2, gives an informal specification in section 3 and shows how it can be implemented using SWORD in section 4. Conclusions are drawn in section 5.

2. THE MILITARY AIRLIFT COMMAND EXAMPLE

The MACE system is used to record details about missions, whose various aspects may be classified differently. An added complication is that sometimes the simple fact that an aspect of a mission is highly classified, is itself classified. Those people with low clearances obviously cannot be allowed to see details about a highly classified aspect of a mission. However, being denied such knowledge informs the person that the aspect is highly classified, which is in itself classified information. Thus the person cannot be denied access, otherwise they learn something they are not cleared to know. A system which does not allow people to access something, at the same time as insisting that they are allowed access, is inconsistent and so cannot be built.

The method for overcoming the potential inconsistency, that has been adopted by MACE, is to employ cover stories. A cover story is false, or less accurate, information that is shown to people with low clearances in place of some highly classified information, where that classification is itself classified high.

For example, suppose the plans for a Troop Carrier are Secret. However, the fact that the plans are classified Secret is in itself Secret information. That is, a person with a clearance of Secret is able to determine that the plans are

¹Those secure DBMS products based on classified views, eg. TRUDATA [Knode88], can be used without having to polyinstantiate [Wilson88].

classified Secret, and can observe their contents. However, a person with a clearance lower than Secret is not even able to know that the plans are Secret, let alone see what they contain.

The problem arises when a person with a low clearance asks to see the plans. If they are told that they cannot see them, they might then be able to deduce that this is because the plans are classified Secret. This, however, constitutes the discovery of information classified higher than their clearance. The problem can be avoided by adopting a cover story. In this case, a suitable cover story for the Troop Carrier might be that the plans are for an enormous ornamental Wooden Horse. This "plan" would be Unclassified and shown to anyone, with a clearance less than Secret, who asks to see the plans.

MACE has further requirements regarding the system's integrity, in particular the specification identifies various states that the system should never enter. Many of these constraints are based on the classification of information held within the DBMS. For example, in MACE, the fact that a particular aircraft is assigned to some mission must always be unclassified, even though details about the mission or the assignment may be classified higher.

From the detailed requirements for USTRANSCOM/MAC, it is concluded [Nelson91] that the DBMS must:

- Support cover stories,
- Provide data element labels at the human and program interfaces,
- Define and enforce classification constraints,
- Provide a single composite view of data to cleared personnel.

Similarly, for MACE we require:

- support for cover stories, to prevent the fact that certain information is highly classified from being revealed to clients with low clearances,
- a relational database with fields individually classified,
- the ability to define constraints based on the classification and value of fields,
- the DBMS to have consistent and well defined semantics for all clients, regardless of clearance.

3. THE MACE SPECIFICATION

In this section, an informal specification of MACE is given. The notation used is that of [Sell92], which is a graphical representation of SPEAR [Wiseman91c], an extended entity-relationship model that can be used to describe confidentiality controls. A summary of the notation is given in Appendix A.

3.1 The basic data structures for MACE

The MACE system requires information to be held about Aircraft, Missions and the Assignment of Aircraft to Missions. This high level requirement is illustrated in Figure 3.1a.

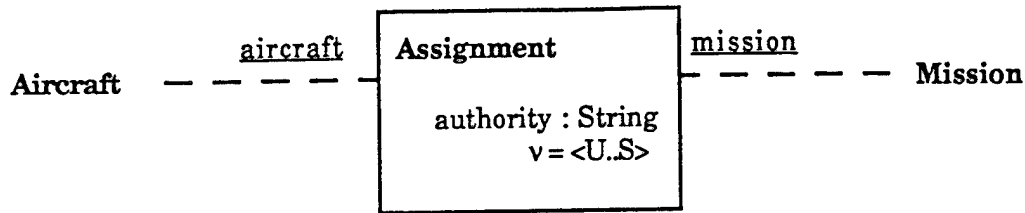


Figure 3.1a: The Basic MACE Requirement

Each assignment relates one aircraft and one mission, but no aircraft may be assigned to more than one mission and no mission may be assigned more than one aircraft. Note that not all aircraft need be assigned to a mission and not all missions need have an aircraft assigned to them. The authority under which an assignment is made is recorded with each assignment. This is a single value of type String.

The authority of an assignment is given a classification in the range Unclassified to Secret. However the fact that an assignment has an authority is always Unclassified. All other aspects of an assignment are also Unclassified.

Figure 3.1b shows that each aircraft has an identity and a type, but no two aircraft can ever have the same identity. Each aircraft is either stationed on the ground at some location or is airborne.

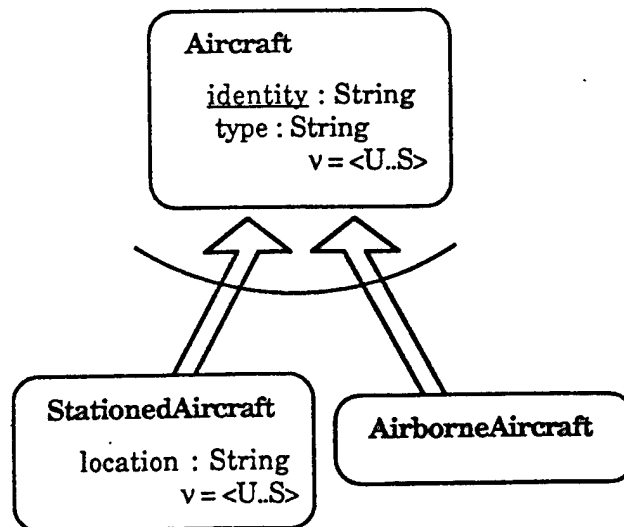


Figure 3.1b: Aircraft Information.

The type of an aircraft is classified in the range Unclassified to Secret, while all other aspects remain Unclassified. Similarly, the location of a

stationed aircraft is classified between Unclassified and Secret. A subtle point which is of particular note, is that, in MACE, the existence of a stationed aircraft or of an airborne aircraft is always Unclassified. This means it will always be possible to discover whether an aircraft is stationed or airborne. That is, the 'state' of an aircraft is always Unclassified.

Missions, as Figure 3.1c shows, are similar to Aircraft in that they have an identity and a destination, but no two missions have the same identity. The destination is where the aircraft flying the mission will ultimately end up (assuming all goes well). Missions are either attack missions or convoy missions, but not both. An attack mission has a target and a time-over-target (tot). A convoy mission is when an aircraft is simply flying from one place to another and so has no extra attributes.

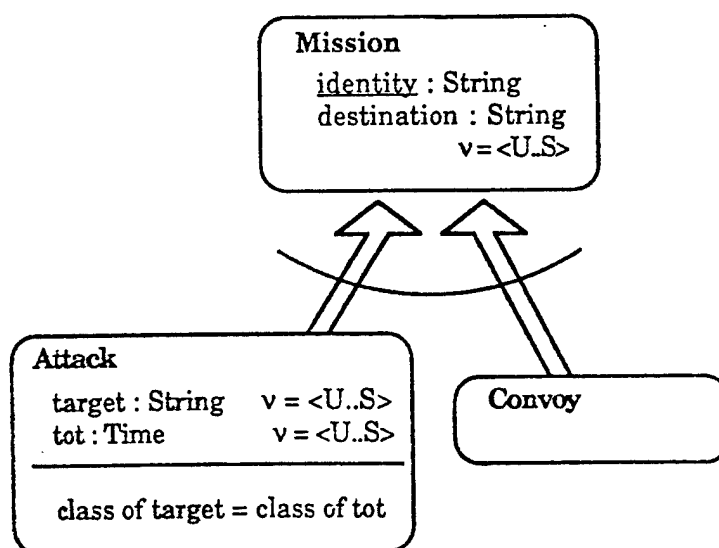


Figure 3.1c: Mission Information.

The attack class is an example of a class constrained by an additional predicate. Beneath the line is a condition which insists that the target and time-over-target of all attacks are classified the same. This is called a Uniform Classification Constraint and is introduced here as an example of the kind of extra constraint that could be included in a real application.

Figure 3.1d provides more details about assignments. It shows that an assignment is either active, in that the assigned aircraft is in the air, or is pending, because the aircraft has not yet taken off. An active assignment has an estimated time of arrival (eta), which is the time the assignment is expected to be complete. A pending assignment has a departure time and a flight time, which are estimates of the time it will become active and of its duration.

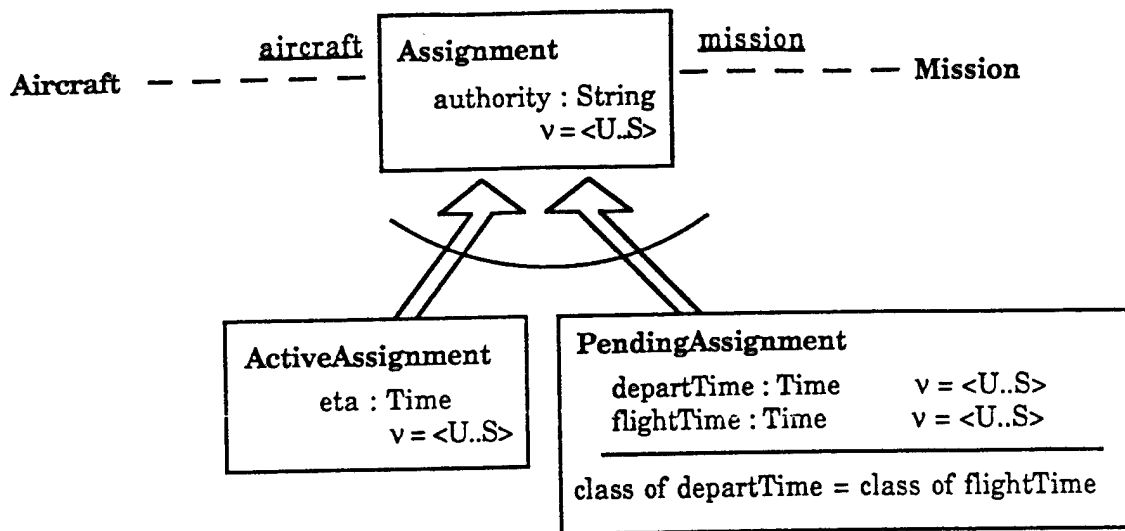


Figure 3.1d: Assignment Information.

In Figure 3.1e, missions are split three ways, into active missions, pending missions and, by virtue of the dashed arrows, others. The intent is that active missions are those assigned to active assignments and pending missions are those assigned to pending assignments. Since not all missions are assigned, some missions will be neither active nor pending. This intent, however, is not expressed in Figure 3.1e but in Figure 3.1f.

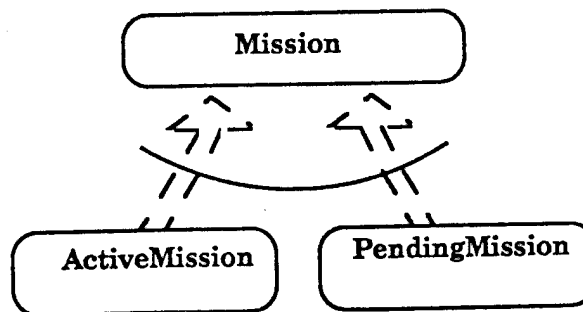


Figure 3.1e: Missions may be active, pending or neither.

In Figure 3.1f, details of the constraints on assignments are shown in terms of the new classes of mission introduced in Figure 3.1d. An active assignment relates an airborne aircraft and an active mission, moreover all airborne aircraft and active missions are related by an active assignment. Also, if a stationed aircraft is assigned to a mission, that mission must be pending.

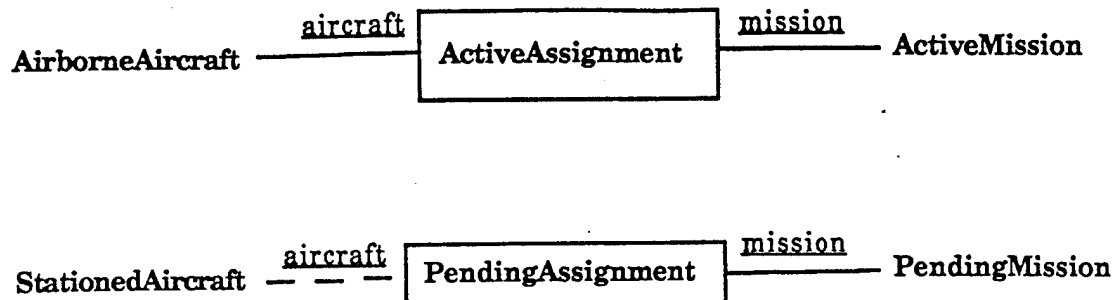


Figure 3.1f: Assignment Constraint Details.

This concludes the basic description of the MACE requirement. Although section 4.1 shows how this can be implemented using SWORD, a more interesting question is how does the abstract description change in order to accommodate the requirement for cover stories? This is shown in the next section.

3.2 The requirement for cover stories in MACE

In many cases, the fact that the destination of a mission is Secret can be revealed to clients with low clearances. That is, although a client with a low clearance is unable to determine what the destination is, they can be told that access is denied because the details are Secret.

However, in some cases, the very fact that the destination is Secret may itself be Secret information. For example, suppose that an uncleared client knows that a destination is invariably made Secret only if it is Ar Riyad. Then, being told the classification of a destination is Secret suggests that it is very likely to be Ar Riyad. For this reason it is necessary to prevent the uncleared client knowing that the destination is Secret. The only way this can be done is to offer the uncleared client an alternative answer, which is either false or a half truth. That is, the application must provide a plausible cover story to hide the truth.

The cover story for the destination of a mission is effectively more information about a mission. It is false information, in that it is not where the mission is heading. However, it is also true information, in that it is what uncleared clients are told about the mission. Thus the cover story is just as much an observable property of a mission as the true story, so it must be modelled in the same way. In effect, the cover story is 'virtual reality'.

Figure 3.2a shows how Missions must be altered to accommodate a cover story for the destination. A mission still requires a unique identity, but instead of destination there is now an apparent destination (*appDest*). Some missions have a cover story for their destination, in which case an actual mission records the actual destination (*actDest*) of the mission.

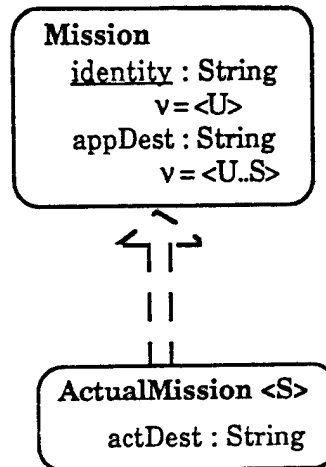


Figure 3.2a: Providing a Cover Story for a Mission's Destination.

When a mission has no cover story for its destination, the destination is held as the apparent destination and the mission has no corresponding actual mission. When a cover story is required, the actual destination of the actual mission gives the true destination and the cover story is in the apparent destination of the corresponding mission.

The existence of an actual mission is always classified Secret, thus clients with lower clearances can never distinguish between a mission with a cover story and one without, since they are unable to tie up a mission with an actual mission.

The requirement for providing cover stories for the destination of a mission is an example of providing a cover story for an attribute. This is relatively straightforward to describe. However, it is sometimes also necessary to provide a cover story to hide the true nature of a mission. This is more difficult to describe since a mission's nature is not seen as an attribute of a mission. Instead, a mission's nature is determined by whether it is also in class Attack or Convoy.

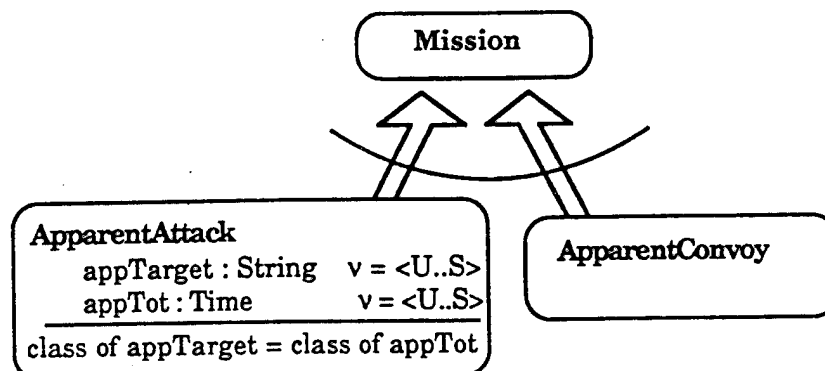


Figure 3.2b: Missions have an apparent nature.

To provide cover stories it is necessary to have an apparent nature for the mission as well as an actual nature if there is indeed a cover story. Figure 3.2b shows that a mission is either apparently an attack or apparently a convoy. This is essentially the same diagram as Figure 3.1c, except the names have changed. Note that, the apparent nature of a mission is always Unclassified, because the existence of apparent attacks and convoys is always Unclassified, and so a mission can always be identified with its corresponding apparent attack or apparent convoy.

Figure 3.2c shows that a mission may also actually be an attack or a convoy, though it could be neither in cases where the actual mission is the same as the apparent mission. While for the apparent nature of a mission it was acceptable for the mission's nature to be Unclassified, this is not the case for the actual nature of a mission. Here it is necessary to make the nature Secret, otherwise clients with lower clearances will know when the apparent information they can see is false.

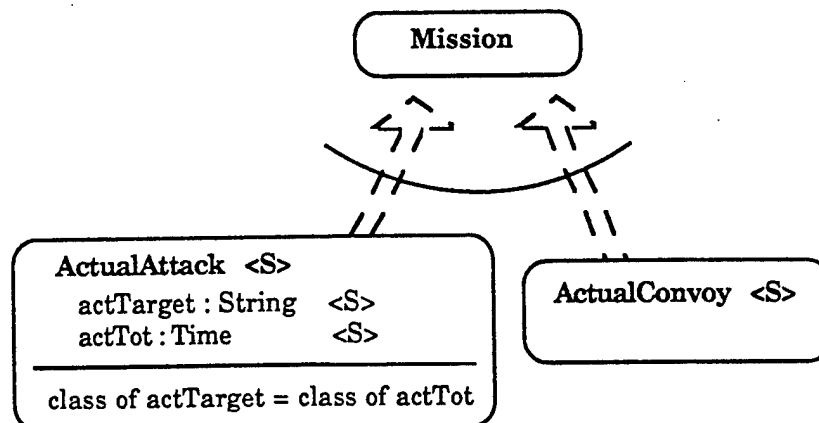


Figure 3.2c: Missions may have an actual nature different to the apparent one.

Thus, as Figure 3.2c shows, the existence of actual attacks and actual convoys is classified Secret. This prevents clients with low clearances from deducing a mission's true nature.

The broad requirement laid down by the owners of the MACE system was to have cover stories for the destination and nature of a mission. However, as all good children know "lies are easily recognised" and so a little lie keeps on growing [Collodi83]. In this case, telling lies about the nature of a mission makes it necessary to tell lies about the flight time of an assignment, because flight time will be dependent on the mission's destination and target.

Thus it is necessary to provide a cover story for the estimated flight time of a pending assignment. This requirement is shown in Figure 3.2d, and is structured in a similar fashion to missions in Figure 3.2a.

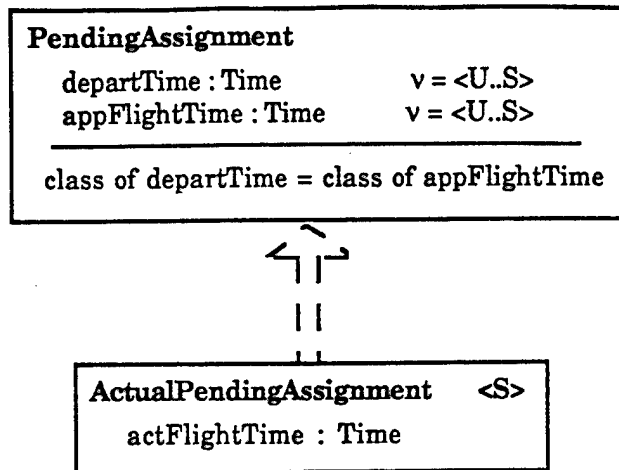


Figure 3.2d: The Flight Time of Pending Assignments must also have a Cover Story.

A person can only sensibly assign an aircraft to a mission if they are fully aware of the mission's details. This means the person must have a clearance of at least Secret. However, it would be desirable to ensure that the person does not make obvious mistakes. To this end a constraint is added which ensures that an assignment has a cover story for the flight time if, and only if, the mission has a cover story for its destination or nature.

To express this constraint it is first necessary to define a covert mission as being one which has one or more cover stories for its various aspects. This is expressed in Figure 3.2e.

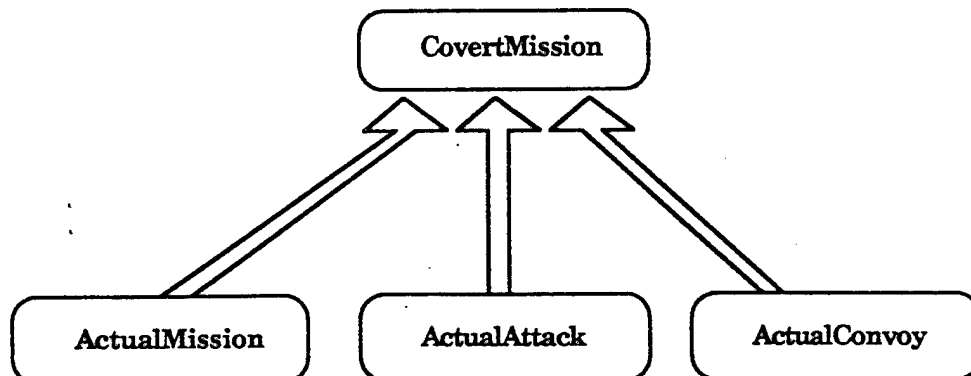


Figure 3.2e: Covert Missions are those with Cover Stories.

Then, covert missions are divided into those that are pending, those that are active and those which are neither. This is shown in Figure 3.2f.

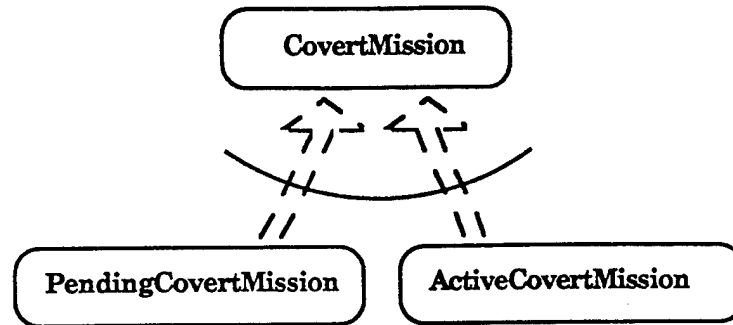


Figure 3.2f: Covert Missions may be Pending or Active.

The diagram in Figure 3.2g then specifies the constraint that is required. That is, all pending covert missions must 'belong' to a pending assignment that has a cover story for its flight time (ie. an assignment that is pending and which has an actual value for its flight time).

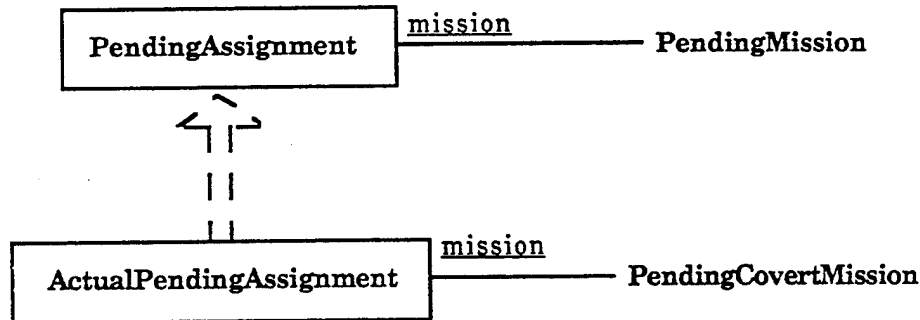


Figure 3.2g: Constraints on Cover Stories.

This completes the modification of the MACE specification to accommodate cover stories about missions. The changes to the diagrams are not inconsequential but, at least in this simple example, are tolerable. While it is accepted that 'syntactic sugar' in the diagrams could be added to make cover stories more palatable, it is felt that the complexity of the specification reflects the true complexity of the requirement.

4. THE SWORD IMPLEMENTATION OF MACE

Having described MACE's requirements at an abstract level using an E-R notation, in this section a more concrete representation, in the form of schemas for SWORD tables, is given.

4.1 The SWORD Tables without cover stories

Each aircraft has two attributes, identity and type. Exactly one value, of type String, is associated with each of these attributes. Also, the identity of an aircraft is unique, in that no two aircraft can have the same identity. Thus we

could represent aircraft as a table with two columns of type String, with a uniqueness constraint on the identity column. The schema for such a table is shown in Figure 4.1a. The schema shows the name and type of each column, plus a range which restricts the classifications of any fields in the column. A column name is underlined to indicate that its values are unique.

Aircraft			
<u>identity</u>	String	[U..U]	
type	String	[U..S]	

Figure 4.1a: First Attempt at a SWORD Schema for Aircraft.

Each aircraft is either stationed or airborne. If the aircraft is stationed it has a location attribute, with which is associated exactly one value of type String. This could be represented by two further tables, as shown in Figure 4.1b. These tables both include an identity column. This is used to equate stationed / airborne aircraft with the 'standard' aircraft details. Referential integrity constraints, given by the \rightarrow symbol, are included to ensure that this standard aircraft always exists.

StationedAircraft			
<u>identity</u>	String	[U..U] \rightarrow Aircraft.identity	
location	String	[U..S]	

AirborneAircraft			
<u>identity</u>	String	[U..U] \rightarrow Aircraft.identity	

Figure 4.1b: First Attempt at SWORD Schemas for Stationed and Airborne Aircraft.

Note, however, that the referential integrity constraints shown in Figure 4.1b are not actually strong enough to enforce all the constraints expressed by the E-R diagram. The proper constraint is that the values in the identity column of StationedAircraft and those in the identity column of AirborneAircraft should partition the values in the identity column of Aircraft. That is:

StationedAircraft.identity *union* AirborneAircraft.identity = Aircraft.identity
 StationedAircraft.identity *intersect* AirborneAircraft.identity = empty

It does not seem appropriate to use three tables, with an inadequate referential integrity constraint to bind them together, since applications are not then prevented from making an aircraft be both stationed and airborne at the same time. It would be possible to specify the constraint properly, using trigger-based application specific integrity checks, but in this case it is simpler to use just one table.

The problem is that not all aircraft are stationed aircraft, in which case they do not have a location. This is solved by allowing the location column to

contain null values. A null location is then taken to mean that the aircraft is not stationed, ie. it is airborne. The resulting table is shown in Figure 4.1c.

Aircraft		
<u>identity</u>	String	[U..U]
type	String	[U..S]
location	String / null	[U..S]

Figure 4.1c: The Final Schema for Aircraft, Airborne Aircraft and Stationed Aircraft.

A table for missions can be constructed in a similar fashion. The attributes for attack missions are represented in the table by two columns. Fields in these columns contain null when the mission is a convoy mission. Note that extra constraints are required to ensure that target and time over target are both null or both non-null. The schema is shown in Figure 4.1d.

Missions		
<u>identity</u>	String	[U..U]
destination	String	[U..S]
target	String / null	[U..S]
tot	Time / null	[U..S]
target is null \Leftrightarrow tot is null		
uniform target, tot		

Figure 4.1d: The Schema for Missions, Attack Missions and Convoy Missions.

Assignments are also mapped to a single table, as described in Figure 4.1e, in a similar way. Here, however, columns are also required to record which aircraft and which mission are party to each assignment. Since exactly one aircraft and exactly one mission must be party to each assignment, this is easily achieved using an extra column for each. These columns record the unique identity of the aircraft / mission.

Since no aircraft may be assigned to more than one mission, the corresponding aircraft identity column is made unique. Similarly, a mission may only be assigned one aircraft. Thus the columns representing the identities of the aircraft and mission are both made unique. Note that these columns are unique individually, not as a pair (ie. they are not a "composite key"). This is shown in the schema by underlining the column names separately. Also, only existing aircraft and missions may be assigned, so referential integrity constraints are applied from aircraftIdentity to Aircraft.identity and from missionIdentity to Missions.identity.

Assignment		
aircraftIdentity	String	[U..U] → Aircraft.identity
missionIdentity	String	[U..U] → Missions.identity
authority	String	[U..S]
eta	Time / null	[U..S]
departTime	Time / null	[U..S]
flightTime	Time / null	[U..S]
 <u>aircraftIdentity</u>		
<u>missionIdentity</u>		
 departTime is null ⇔ flightTime is null		
eta is null ⇔ departTime is not null		
 uniform departTime, flightTime		

Figure 4.1e: The Schema for Assignments, including Active and Pending.

Each assignment is either an active assignment or a pending assignment, so either one set of attributes or the other are required, but not both. Thus constraints are applied to ensure that one set of columns or the other, but not both, contain null values.

4.2 The SWORD tables with cover stories

With the introduction of cover stories, the schemas must account for the extra attributes that are used to represent the apparent and actual attributes of a mission. First, consider the changes necessary to the Missions schema. The new schema is shown in Figure 4.2a. This describes the implementation of missions, attack missions, convoy missions and any associated cover stories.

Each mission has an apparent destination, which has exactly one value of type String associated with it. The actual destination of a mission is also a String, but it is necessary to record when there is no cover story for the destination. This is achieved by allowing null values in the actDest column.

The way the apparent nature of a mission is represented is not changed by the introduction of cover stories. A mission is either apparently an attack, in which case the appTarget and appTot fields are non-null, or it is apparently a convoy, in which case the two columns are null.

The representation of the actual nature of a mission is more complicated, since it is possible that a mission is neither an actual attack or an actual convoy (this is when there is no cover story for the nature of a mission). The problem arises because no attributes are associated with actual convoys and hence 'null' cannot be used to represent their absence. This is overcome by introducing a column to represent the existence of an actual convoy, even though it has no attributes that need representing.

Thus the existence of a cover story in the form of a convoy is recorded by a column of data type Monolean. The type Monolean, which is unique to

SWORD, is strange in that only one value, called Void, has this type. The fields in the column actConvoy therefore contain either the value Void or a null. Note that, although this is equivalent to storing a Boolean without nulls, the use of a Monolean and a null is preferred as it is more symmetric

Missions			
identity	String	[U..U]	
appDest	String	[U..S]	
actDest	String / null	[S..S]	
appTarget	String / null	[U..S]	
appTot	Time / null	[U..S]	
actTarget	String / null	[S..S]	
actTot	Time / null	[S..S]	
actConvoy	Monolean / null	[S..S]	
<p>appTarget is null \Leftrightarrow appTot is null actTarget is null \Leftrightarrow actTot is null actConvoy is not null \Leftrightarrow actTarget is null</p> <p>uniform appTarget, appTot</p>			

Figure 4.2a: The Schema for Missions with Cover Stories.

One requirement expressed by Figure 3.2c is that the actual nature of a mission is classified Secret. One possible way of meeting this requirement would be to classify at Secret the existence of the columns relating to a mission's actual details. While this is possible in SWORD, in this simple case it is not necessary. This is because MACE is admitting to one and all, that clients with clearances lower than Secret may be given a cover story instead of the truth. More demanding applications may wish to keep even this information Secret¹, in which case the existence of the columns would need to be classified Secret and more care would need to be taken in choosing the names of those columns visible to all.

By arranging that the fields containing the actual details are always classified Secret, whether or not they contain null data, clients with low clearances are unable to determine the true nature of a mission. Thus the schema described in Figure 4.2a applies field classification constraints to the columns actTarget and actTot which ensure that all fields within them are classified Secret.

Now consider how the schema for Assignments must be changed. A first attempt at a new schema is shown in Figure 4.2b. Instead of a single flight time column, there are now two columns, one for the apparent flight time and one for the actual flight time if there is a cover story. The constraints governing the use of nulls ensure that there can only be a cover story for the

¹It may be that USTRANSCOM/MAC actually does have a requirement for hiding the fact that certain kinds of cover stories might be employed. However, if this is the case, then by its very nature such information will not be revealed in an open conference.

flight time if the assignment is pending (ie. it has an apparent flight time). The check constraint ensures that, if the assignment is pending, the estimated flight time and the corresponding mission details either both have cover stories or both do not have cover stories.

Assignment		
aircraftIdentity	String	[U..U] → Aircraft.identity
missionIdentity	String	[U..U] → Missions.identity
authority	String	[U..S]
eta	Time / null	[U..S]
departTime	Time / null	[U..S]
appFlightTime	Time / null	[U..S]
actFlightTime	Time / null	[S..S]

<u>aircraftIdentity</u>
<u>missionIdentity</u>

departTime is null ⇔ appFlightTime is null
eta is not null ⇔ departTime is null
actFlightTime is not null ⇒ appFlightTime is not null

uniform departTime, appFlightTime

CHECK appFlightTime is not null ⇒
actFlightTime is not null
⇔ actDest is not null
or actTarget is not null
or actConvoy is not null

FROM Missions WHERE identity = missionIdentity

Figure 4.2b: A First Attempt at a Schema for Assignments with Cover Stories.

The schema shown in Figure 4.2b adequately describes the structure of the data and the constraints that must be imposed. However, the constraints are such that a client with a low clearance is unable to create a new pending assignment. This is because, with the apparent flight time containing a non-null value, the state of the cover stories must be checked to establish their consistency. Unfortunately these details are all classified Secret. A Secret client would be able to check the constraint, but is unable to create a new assignment because the existence of an assignment is constrained to be Unclassified. Thus it would not be possible to create new pending assignments.

Clearly, in the MACE system, an assignment can only be made by a person with a Secret clearance. This is because the person must be able to take into account any cover stories about the mission. So the problem is solved by having the person log-in as an Unclassified client to create the assignment and as a Secret client to update the assignment's cover story information, and allowing the constraint to be violated in between.

To indicate that the constraint may be 'temporarily' violated, a special value is used for the actual flight time, which is neither a Time nor a null. SWORD supports such requirements by having two kinds of data, Dinary and Sterling. Dinary data is used to indicate that the Data Is Not Available or Ready Yet, while Sterling data is of "thoroughly good character". Thus generally, a value in a field is either a null, a Dinary value of some type or a Sterling value of some type.

The data in a column of a table is constrained to be of one type if it is Sterling and of another type (or the same) if it is Dinary. This is shown in the schema diagrams by having two types alongside a column name, in the format 'sterling / dinary'. It is also possible to exclude Dinary values altogether, which is the norm, by just giving the sterling type.

In Figure 4.2c, it is stated that any Sterling values in the actFlightTime column must be of type Time and any Dinary values must be of type Monolean, that is they must be the value Void. The actFlightTime column may also contain nulls.

Assignment		
aircraftIdentity	String [U..U]	→ Aircraft.identity
missionIdentity	String [U..U]	→ Missions.identity
authority	String [U..S]	
eta	Time / null	[U..S]
departTime	Time / null	[U..S]
appFlightTime	Time / null	[U..S]
actFlightTime	Time / Monolean / null	[S..S]
<u>aircraftIdentity</u>		
<u>missionIdentity</u>		
departTime is null ⇔ appFlightTime is null		
eta is not null ⇔ departTime is null		
actFlightTime is not null ⇒ appFlightTime is not null		
uniform departTime, appFlightTime		
CHECK appFlightTime is not null ⇒		
actFlightTime is not null		
and actFlightTime <> void		
⇒ actDest is not null		
or actTarget is not null		
or actConvoy is not null		
actFlightTime is not null		
or actFlightTime = void		
⇐ actDest is not null		
or actTarget is not null		
or actConvoy is not null		
FROM Missions WHERE identity = missionIdentity		

Figure 4.2c: The Schema for Assignments with Cover Stories.

Compared to the first attempt in Figure 4.2b, the final schema for assignments contains a weaker check constraint. This effectively allows a Dinary void in the actual flight time to represent "don't know if there is a cover story". Now the person can log-in as an Unclassified client and create a new pending assignment using void for the actual flight time. The check constraint is trivially true, so there is no need for the Unclassified client to observe the contents of Secret fields.

Once the assignment has been created, the person can log-in as a Secret client and update the actual flight time to be a null if the mission has no cover story or to an actual time if it has. Note, however, that the constraints do not force the person to complete this second part of the task.

4.3 Examples

The table in Figure 4.3a shows five example missions, each with different "cover story" characteristics.

Missions										
identity	appDest		actDest	appTarget		appTot	actTarget	actTot	actConvoy	
[U]	[U..S]		[S]	[U..S]		[U..S]	[S]	[S]	[S]	
"M42"	"Riyad"	[U]	Null	Null	[U]	Null [U]	Null	Null	Null	
"M101"	"Dhahran"	[C]	Null	"Basrah"	[S]	23:40 [S]	Null	Null	Null	
"M7x5"	"Dhahran"	[U]	Null	"Iraq"	[U]	07:30 [U]	"Basrah"	07:20	Null	
"M17b"	"Dharhan"	[U]	"Riyad"	Null	[U]	Null [U]	"Baghdad"	14:40	Null	
"M6z7"	"Riyad"	[U]	"895269"	"Iraq"	[C]	22:30 [C]	Null	Null	Void	

Figure 4.3a: Some Example Missions.

M42 is a simple convoy mission to Riyadh and there is no cover story for either its destination or nature. Of course, those clients with Unclassified or Confidential clearances do not know this.

M101 is another mission which does not involve a cover story. However, this mission is sensitive in that the destination and target are classified. A client with a clearance of Unclassified is able to determine that there is a mission called M101, but is unable to ascertain any details.

The actual mission details of M7x5 are quite sensitive, but it is necessary to reveal something of its nature to clients with clearances lower than Secret. Thus a cover story is provided to give these clients a "sanitised" version of the details. The mission is actually an attack on Basrah at 07:20. However, to those with a low clearance the information is much less specific, stating that the mission is to attack Iraq at around 07:30.

A full scale lie is being perpetrated for M17b. Here the mission is actually an attack on Baghdad at 14:40, with the aircraft returning to Riyadh. However, those with low clearances see a convoy mission which takes the aircraft to Dharhan.

Mission M6z7 is interesting since it involves a lie to the Confidential clients which is being hidden from Unclassified clients. The mission is actually a

convoy mission to map reference 895269, but this is a particularly sensitive location. To hide this the cover story is that it is on a sensitive mission to attack Iraq and will then return to Riyadh. Note here the use of Void to indicate that, while the mission is actually a convoy, a cover story is provided to hide this.

Figures 4.3b and 4.3c show the result of the query.....

SELECT * FROM Missions

.....when issued by clients with clearances of Unclassified and Confidential respectively. Note that, as Figure 4.3b shows, clients with a clearance of Unclassified are able to determine the classifications of all the fields. This functionality does not prevent SWORD exhibiting Information Flow Security, because the Insert Low approach is adopted [Wiseman90]. There is, of course, nothing to stop the application from further restricting what a client can observe by applying other security controls, as illustrated in section 4.4.

identity [U]	appDest [U..S]	actDest [S]	appTarget [U..S]	appTot [U..S]	actTarget [S]	actTot [S]	actConvoy [S]
"M42"	"Riyad"	[U]	Null	[U]	Null	[U]	
"M101"		[C]		[S]		[S]	
"M7x5"	"Dhahran"	[U]	"Iraq"	[U]	07:30	[U]	
"M17b"	"Dharhan"	[U]	Null	[U]	Null	[U]	
"M6z7"	"Riyad"	[U]		[C]		[C]	

Figure 4.3b: Missions as Seen by Clients Cleared to Unclassified.

As Figure 4.3c shows, a client with a clearance of Confidential is not able to see much more than a client with a clearance of Unclassified. It does, however, serve to show that the solution works for a general hierarchy of classifications, not just for two.

identity [U]	appDest [U..S]	actDest [S]	appTarget [U..S]	appTot [U..S]	actTarget [S]	actTot [S]	actConvoy [S]
"M42"	"Riyad"	[U]	Null	[U]	Null	[U]	
"M101"	"Dhahran"	[C]		[S]		[S]	
"M7x5"	"Dhahran"	[U]	"Iraq"	[U]	07:30	[U]	
"M17b"	"Dharhan"	[U]	Null	[U]	Null	[U]	
"M6z7"	"Riyad"	[U]	"Iraq"	[C]	22:30	[C]	

Figure 4.3c: Missions as Seen by Clients Cleared to Confidential.

4.4 Dynamic checks

The specification of MACE given in section 3.2 only gives the requirements for the static structure of the application. Dynamic aspects, such as the circumstances under which a person may create a new mission, are missing and this is a shortcoming of modelling at the E-R level.

One dynamic requirement which is of particular concern in MACE, is that only users with high clearances may add a cover story to a mission. Unfortunately, the schemas described in section 4.2 do nothing to prevent a low user creating a new mission with a cover story, or updating an existing

mission's cover story. This is because SWORD allows clients to create and alter fields whose classification dominates the client's clearance (ie. SWORD supports the general case).

The required checks can, however, be added to the database schema by using SWORD's trigger mechanism. The required trigger definitions are shown in Figure 4.4a.

The trigger module changeCover, which is attached to the Missions table, defines two triggers. One fires whenever a new row is inserted into Missions and the other fires whenever Missions is updated.

When a new row is inserted, which is something that can only be done by a client whose clearance is Unclassified, the actDest field is checked. If it is not null, an exception is raised which abandons the query. Thus, when a mission is created, there must be no cover story for the destination.

```
MODULE changeCover
  ON INSERT INTO Missions
  TRIGGER
    IF POSSIBLY null <> ( ALL SELECT actDest FROM new)
    THEN
      RAISE "Cannot add cover story"
    FI
  END

  ON UPDATE Missions
  TRIGGER
    IF clearance <> [Secret]
    AND MODIFIED actDest
    THEN
      RAISE "Cannot update cover story"
    FI
  END
```

Figure 4.4a: Extra Constraints on Modifying Missions.

When a row is updated, the clearance of the client is checked. If the client's clearance is not Secret and the update is setting the value in the actDest column, an exception is raised which aborts the update query. Thus only clients with a clearance of Secret can update the cover story of a mission's destination.

Note that no trigger is defined for a delete operation. This is because a client with a clearance of Unclassified is deemed free to delete a mission, including any attached cover story information. However, further triggers could be provided to cater for different requirements.

```

MODULE controlAssignments
  ON INSERT INTO Assignment
  TRIGGER
    IF POSSIBLY user <> ( SOME SELECT authority FROM new)
    THEN
      RAISE "Imposter!"
    FI
  END

  ON UPDATE Assignment
  TRIGGER
    IF MODIFIED authority
    THEN
      RAISE "Cannot change authority"
    FI
  END

  ON DELETE FROM Assignment
  TRIGGER
    IF POSSIBLY user <> (SOME SELECT authority FROM chosen)
    THEN
      RAISE "Not your Assignment"
    FI
  END

```

Figure 4.4b: Constraints on Modifying Assignments.

Another example of the use of triggers to add further application specific controls to the database is shown in Figure 4.4b. This is intended to ensure that only the user who made an assignment is able to delete the assignment.

A row may only be inserted if the authority gives the name of the user on whose behalf the query is made. Subsequently, the authority of an assignment may not be updated. Thus when an attempt is made to delete a row, the authority field gives the name of the user who made assignment. The trigger raises an exception if the client is not running on behalf of the user that made the assignment in the first place, or it is not possible to observe the assignment's authority. This latter point poses a particularly awkward problem.

If the authority of an assignment is classified above Unclassified, a client capable of deleting the assignment will be incapable of determining whether they are allowed to delete it. Such a situation would be avoided by allowing the authority to be sanitised under certain circumstances, such as the example shown in Figure 4.4c.

```

MODULE controlAssignments
.....
ON UPDATE Assignment
TRIGGER
  IF MODIFIED authority
  AND clearance NOT DOM (SOME SELECT CLASS OF authority
                        FROM chosen)

  AND NOT EXISTS( SELECT * FROM permissions
                  WHERE name = user AND priv = "ChangeAuthority" )

  THEN
    RAISE "Cannot change authority"
  FI
END
.....

```

Figure 4.4c: Allowing Privilege Users to Change Authority.

This trigger assumes the existence of another table called permissions, which has two columns; user and priv. This gives the list of permissions granted to each user. This table is examined using a select query, to discover whether the user has the privilege to change the authority of a mission that may not belong to them. If not, an exception is raised and the update query is abandoned.

The triggers given here are just examples of what may be achieved. In general the central part of the SWORD DBMS imposes no more than the bare minimum of constraints on the application to ensure that information flow security is upheld. Elementary constraints, such as typing and uniqueness, may be applied as required. Further constraints are then added using the trigger mechanism. This approach avoids the problem of providing constraints in the DBMS which are suitable for one application, but which cause others severe problems and result in costly work-arounds.

4.5 Views

The inclusion of the extra columns to hold the cover story information is somewhat inconvenient, since the absence or presence of a cover story is given by whether one field or another is null. This gives particular problems to application software which is intended to be run by clients with different clearances. It would be more convenient if the software could just ask for "the destination", and receive the apparent destination if the clients clearance is low or there is no cover story, and receive the actual destination otherwise.

This ability to "disguise" the columns of the table is provided by views, which in SWORD are implemented using the general purpose trigger mechanism. When a view is observed, each row of the base table is taken and a new row is computed based on some given functions. This new row is discarded if it does not meet the integrity constraints of the view. When a view is updated, or a new row inserted, the value in each modified field is directed towards one of the columns of the base table. This is given by an expression which is conditional on the values of the inserted or modified row. Note that, if

the client is not cleared to observe the value of a conditional expression, the value acts like false.

Figure 4.5a shows the definition of a view which can be used to ease access to the information about missions. The view has six columns, the first four of which are straightforward and give the mission's identity, destination, target and time over target. The last two columns, which are of type Boolean, indicate whether the mission has a cover story in place for its destination or nature.

AirMissions based on Missions			
<u>identity</u>	String	[U..U]	
	←	identity	
	→	identity	
destination	String	[U..S]	
	←	CASE WHEN actDest is not null	
		THEN actDest ELSE appDest END	
	→	CASE WHEN coverDest	
		THEN actDest ELSE appDest END	
target	String / null	[U..S]	
	←	CASE WHEN actTarget is not null	
		THEN actTarget ELSE appTarget END	
	→	CASE WHEN coverNature	
		THEN actTarget ELSE appTarget END	
tot	Time / null	[U..S]	
	←	CASE WHEN actTot is not null	
		THEN actTot ELSE appTot END	
	→	CASE WHEN coverNature	
		THEN actTot ELSE appTot END	
coverDest	Boolean	[S..S]	
	←	actDest is not null	
	→		
coverNature	Boolean	[S..S]	
	←	actTarget is not null or actTot is not null	
		or actConvoy is not null	
	→		
tot is null ⇔ target is null			
uniform target, tot			

Figure 4.5a: Providing a View on Missions.

The Boolean type is an unusual one to find in a Relational DBMS, but is provided by SWORD so that the standard relational database normalisation process can be halted early. This is essential because normalisation can introduce integrity checks which often cannot be evaluated in the face of the confidentiality controls in a secure database.

Of course, clients with clearances lower than Secret are unable to determine whether some aspect of a mission has a cover story. This is because the fields containing the cover stories are always classified Secret, and thus fields in the two Boolean columns will always be labelled Secret.

Figure 4.5b shows the result of a client with a clearance of Unclassified selecting all data from the view, given the underlying table is as shown in Figure 4.3a. Note that none of the Boolean values are visible, so the client is unable to determine which missions have cover stories.

identity [U]	destination [U..S]		target [U..S]		tot [U..S]		coverDest [S]	coverNature [S]
"M42"	"Riyad"	[U]	Null	[U]	Null	[U]
"M101"	[C]	[S]	[S]
"M7x5"	"Dhahran"	[U]	"Iraq"	[U]	07:30	[U]
"M17b"	"Dhahran"	[U]	Null	[U]	Null	[U]
"M6z7"	"Riyad"	[U]	[C]	[C]

Figure 4.5b: AirMissions as Seen by Clients with Unclassified Clearance.

Figure 4.5c shows the result for clients with Secret clearances. Here it is possible to see clearly which missions have cover stories for their destination or their nature. The destination and target information presented reflects whether there is a cover story, in that these clients always see the 'true' information.

identity [U]	destination [U..S]		target [U..S]		tot [U..S]		coverDest [S]	coverNature [S]
"M42"	"Riyad"	[U]	Null	[U]	Null	[U]	False	False
"M101"	"Dhahran"	[C]	"Basrah"	[S]	23:40	[S]	False	False
"M7x5"	"Dhahran"	[U]	"Basrah"	[S]	07:20	[S]	False	True
"M17b"	"Riyad"	[S]	"Baghdad"	[S]	14:40	[S]	True	True
"M6z7"	"895269"	[S]	Null	[S]	Null	[S]	True	True

Figure 4.5c: AirMissions as Seen by Clients with Secret Clearance.

The purpose of the two Boolean columns is to indicate to clients with Secret clearances when clients of lower clearance are being misled about mission details. Numerous other views are possible, and which is appropriate would depend on exactly how the people would use the MACE system. One potentially useful variant would be to include the actual cover story information rather than just a Boolean flag.

The Boolean flags also play an important role in maintaining the database, in that they provide a means by which a client cleared to Secret can add a cover story without having to resort to the underlying table.

For example, suppose a Secret client wanted to make the fact that M42 was on a convoy mission to Riyadh a cover story, by making its actual mission an attack on Baghdad at 10:15. The following query would not achieve this.....

UPDATE AirMissions SET target = "Riyad", tot = 10:15 WHERE identity = "M42"

.....because this would be an attempt to update the unclassified nulls in the apparent target and tot fields, which would constitute a flow down. Instead, the following query must be used.....

```
UPDATE AirMissions SET target = "Riyad", tot = 10:15, coverNature = True
WHERE identity = "M42"
```

.....which indicates the fact that the actual target and tot is to be updated by also updating coverNature to True. Note that this query would also work if it were executed by a client cleared to Unclassified, but this does not constitute a failure of Information Flow Security, because information is 'flowing up'. A trigger similar to that in Figure 4.4a could be defined to prevent such action if required.

If a person wishes to create a new mission and this is to have a cover story, the person must first create the mission by logging in as an Unclassified client. At this point the person would supply all the unclassified information about the mission, including the cover story details. Then, after logging in at Secret, the person would issue a suitable update to enter the true details about the mission.

5. CONCLUSIONS

In this paper it has been shown, by example, how to model a requirement for 'cover stories' using the SPEAR notation and implement it using the SWORD secure DBMS. The example used, MACE, is based on the requirements of the Military Airlift Command project [Nelson91].

A feature common to most application requirements is for a class of entities where no two entities have the same identity. An example of this in MACE is that aircraft have unique names. Another common feature is where an entity has only one value for a particular attribute. For example, in MACE an aircraft has only one type. When implemented in a relational DBMS, these common features equate to uniqueness constraints which prevent a table having two rows with the same values in a 'key' column.

Such constraints are so common in database applications that they are taken for granted by application designers. However, it is unfortunate that most attempts to add security to a DBMS have removed their ability to enforce such constraints. SWORD is a notable exception to this, because it adopts the Insert Low approach [Wiseman90] and so is able to enforce uniqueness constraints without compromising on information flow security.

If the secure DBMS is unable to help the application designer enforce the elementary integrity constraints required by the structure of the application, it is necessary to build extra application specific code to enforce the constraints. This is the approach adopted by the Military Airlift Command project

[Nelson91]. The obvious disadvantage of this approach is the non-recoverable cost of producing and validating the extra software. Another, less obvious, disadvantage is that it is actually quite difficult to provide the integrity checks, especially when consideration is given to the deletion, sanitisation and downgrading of data.

Using a DBMS like SWORD, on the other hand, requires no extra application software to be written and evaluated in order to enforce the basic integrity constraints which are taken for granted by the designers of non-secure applications. This is because all the required integrity enforcement is provided, as it should be, by the DBMS. Thus the use of SWORD would result in cheaper implementations, since a low assurance, 'trusted subject' SWORD DBMS should cost around the same to produce as a DBMS that forces the application to polyinstantiate.

It has been suggested that SWORD is inferior to 'polyinstantiating' DBMSs because it cannot satisfy the requirement for cover stories. Fortunately, as this paper has shown, this is not true. In fact the use of cover stories requires the introduction of severe integrity constraints in order to prevent the 'high' users becoming confused about the true state of the system [Wiseman91a]. The inability of a polyinstantiating DBMS to help enforce such constraints actually makes SWORD more appropriate in such circumstances.

The conclusion is that SWORD is able to support the complex integrity and information flow security requirements of secure database applications, including the need for cover stories. The main reason this is possible is because SWORD does not force the application to polyinstantiate and then provide its own integrity checking software.

7. REFERENCES

- Collodi83 "Pinocchio - Storia di un Burattino", C.Collodi, 1883. Translated into English by M.A.Murray, Publ. Dent 1951.
- Jajodia90 "Polyinstantiation Integrity in Multilevel Relations", S.Jajodia & R.Sandhu, *Procs. 4th IFIP WG11.3 Workshop on Database Security*, Halifax, UK, September 1990.
- Knode88 "Making Databases Secure with TRUDATA Technology", R.B.Knode & R.Hunt, *Procs. 4th Aerospace Computer Security Applications Conference*, Orlando, FL, December 1988, pp82-90.
- Lewis91 "The Front End Approach to Database Security", S.R.Lewis, *Procs. 7th Int. IFIP TC11 Conf. on Information Security*, Brighton, UK, May 1991.

- Nelson91 "Using Polyinstantiation to Develop an MLS Application", D.Nelson & C.Paradise, *Procs. 7th Annual Computer Security Applications Conference*, San Antonio, Texas, December 1991, pp12-22.
- Sell92 "The SPEAR Data Design Method", P.J.Sell, *Procs. IFIP WG11.3 Workshop on Database Security*, Vancouver BC, August 1992.
- Smith89 "Going Beyond Technology to Meet the Challenges of Multilevel Database Security", G.W.Smith, *Procs. 12th National Computer Security Conference*, Baltimore, MD, October 1989, pp.1-10.
- Wilson88 "Views as the Security Objects in a Multilevel Secure Relational DBMS", J.Wilson, *Procs. IEEE Symp. Security and Privacy*, Oakland, CA, April 1988, pp70-84.
- Wiseman90 "Control of Confidentiality in Databases", S.R.Wiseman, *Computers and Security Journal*, Vol. 9, No. 6, October 1990, pp.529 - 537.
- Wiseman91a "Lies, Damned Lies and Databases", S.R.Wiseman, *RSRE Memorandum 4503*, July 1991.
- Wiseman91b "Notes on the Polyinstantiation Problem", S.R.Wiseman, *RSRE Memorandum No. 4504*, July 1991.
- Wiseman91c "Abstract and Concrete Models for Secure Database Applications", S.R.Wiseman, *Procs. of Fifth IFIP WG11.3 Working Conference on Database Security*, Shepherdstown, West Virginia, November 1991.
- Wood92 "The SWORD Multilevel Secure DBMS", A.W.Wood, S.R.Lewis & S.R.Wiseman, *RSRE Report 92005*, February 1992¹.

¹The RSRE Reports and Memos referenced here are freely available on request.

APPENDIX A. A SUMMARY OF THE SPEAR NOTATION

In SPEAR, entities are gathered together into classes. If an entity belongs to a class, the entity must have some values for certain observable attributes. Unless otherwise stated, there must be exactly one value for each attribute. In Figure A1, the class called Ship is shown to require two attributes, called name and tonnage. Entities of class Ship must have exactly one string associated with their name attribute and exactly one integer with their tonnage attribute.

A uniqueness constraint is applied to the name attribute of the class ship. This is shown by underlining the attribute name and means that no two entities in the class Ship may have the same name. By default, the name of a ship must be Unclassified, but the tonnage must be classified in the range given after the v, that is Unclassified to Secret. The existence of each ship is classified between Unclassified and Secret, by virtue of the classification range given to the side of the class's name. If this were omitted, the existence classifications would default to Unclassified.

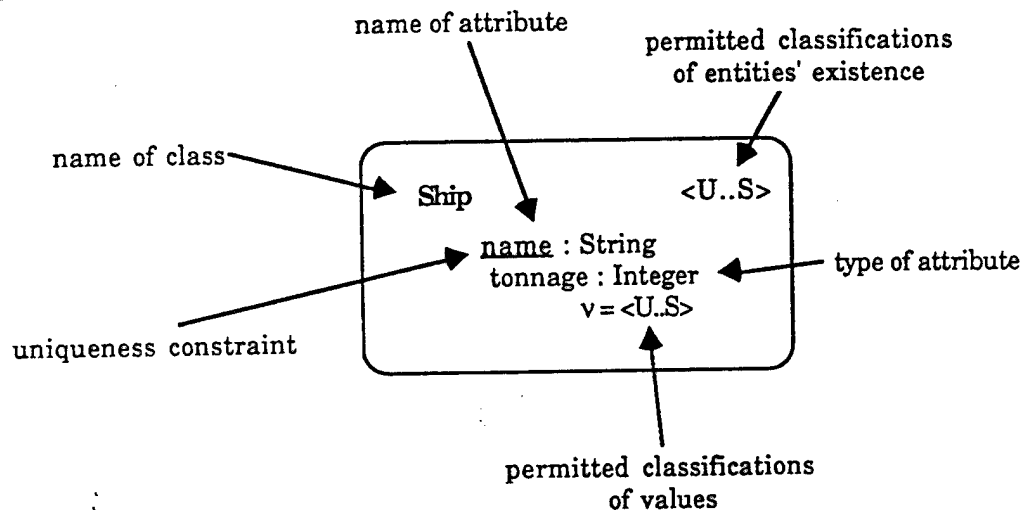


Figure A1. Example SPEAR Diagram for Classes

Relationships in SPEAR have attributes, in the same way that entities have, and parties, which are collections of entities that are party to the relationship. They can be gathered together into families, but in order to belong to a family, a relationship must have certain attributes and certain parties. Figure A2 shows an example of a family, called Commands. Membership of Commands requires that a relationship has an attribute called effective and two parties called craft and captain. Those entities of the craft party must be in the class Ship and those of the captain party must be of class Person.

No two relationships in the Commands family may have the same Ship as their craft party, because the party name is underlined. Similarly, no two relationships in Commands may have the same captain. The solid line for the

craft party indicates that each entity in the Ship class must be the craft party of at least one relationship in Commands. However, the dashed line for the captain party means that not all entities in the Person class need be captains of a ship.

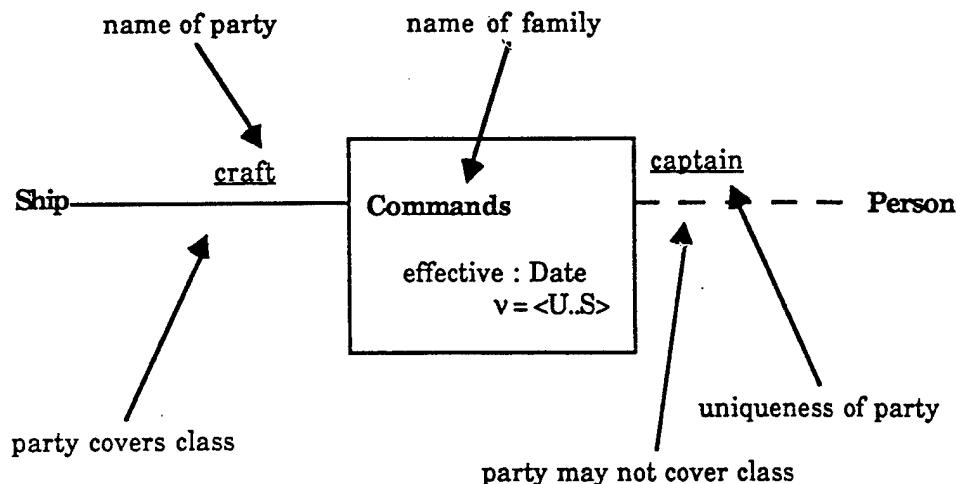


Figure A2. Example SPEAR Diagram for Families.

In Figure A3, the wide arrow states that for an entity to be a member of the class RoyalNavyShip, it must also belong to the class ship. However, because the arrow is dashed, not all ships need be in the Royal Navy.

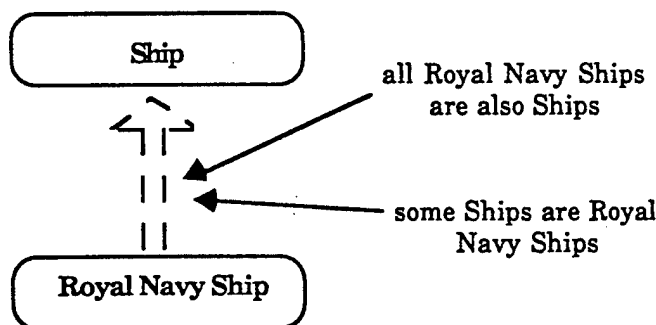


Figure A3. Example SPEAR Diagram for Class Hierarchies.

By making the wide arrow solid, as in Figure A4, entities in the class Ship are required to be in at least one of the classes of ships underway and ships docked. The addition of an arc specifies that no ship can be both underway and docked.

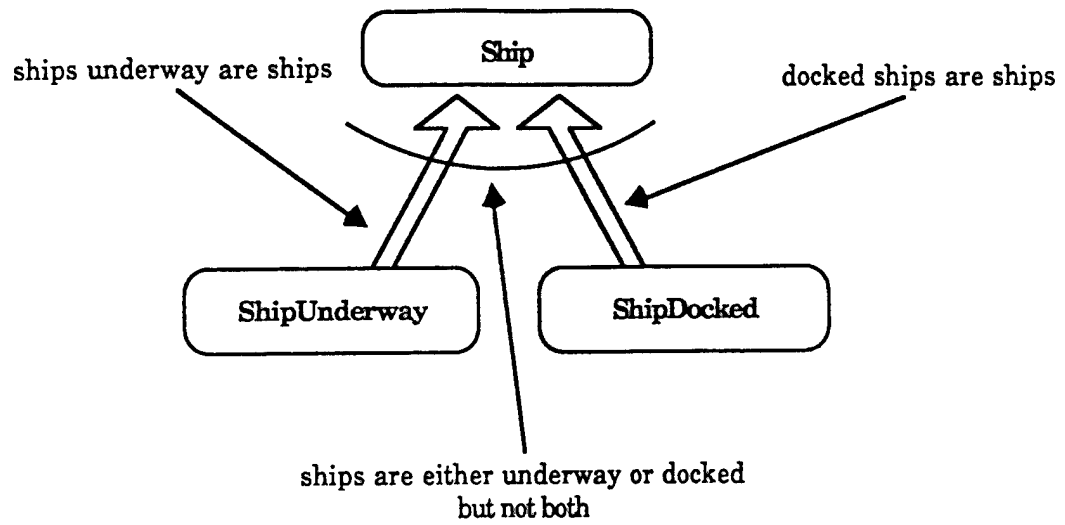


Figure A4. Example SPEAR Diagram for Disjoint Class Hierarchies.

Extending access control with duties realized by active mechanisms

Dirk Jonscher

Universität Rostock
Fachbereich Informatik
A.-Einstein-Str. 21
D - O - 2500 Rostock

Tel.: + 49 - 381 - 44424-156 (or: ... -159)
Fax: + 49 - 381 - 446089

E-mail: jonscher@informatik.uni-rostock.dbp.de

Abstract

Modelling the access behaviour of users is, like database modelling in general, one of the most cumbersome issues in the process of designing and managing a database. So it is worth to develop mechanisms, which are more closely to the part of the real world to be modelled.

Roles describing the functional or organizational position of users in a company are one of these more "semantic" ideas. Roles, however, cannot be an independent concept. While users playing a special role acquire a set of rights to do something, usually they also acquire a *responsibility domain*, i.e. a set of tasks, which they have to fulfil. The latter can be regarded as a kind of *duties* (or obligations), typical of this role and obviously different from permissions to do something, since they carry a normative aspect.

So it would be useful to have a method to describe this difference concerning the access behaviour and therefore the security policy. This paper deals with such an idea, called *duties*, and shows how it can be implemented by virtue of a somewhat new database technology: *active mechanisms* (also denoted as *triggers* or *imperative rules*).

1. Introduction

New directions in database technology force the security community to develop new security policies and access control mechanisms, which must be appropriate to the corresponding data models and should be as "semantic" as the mechanisms of the database system they have to deal with. On the other hand, however, these new technologies also offer new opportunities for access control policies, more closely to the part of the real world to be modelled (in the following referred to as the *Universe of Discourse* - UoD). One of these promising concepts are active mechanisms (cf. /Chak 89/; also denoted as *triggers* - /Eswa 76/, /KoDM 88/ and /Kotz 89/ - or (*imperative*) *rules* - /Ston 90/ and /GaGD 91/). Because of their foundation on situation monitoring, they are very useful for access control, too.

Besides their obvious application of establishing an additional shield around highly sensitive data and checking each access to that data by an "independent" trigger attached to it (i.e. independent of the ordinary access control mechanisms; cf. /Kotz 89/), they are also able to support a new concept of access control, so-called *duties*. Duties describe a responsibility domain of a user, i.e. a set of tasks (as a bundle of actions), which have to be fulfilled under certain circumstances. Some tasks are critical for the smooth operation of a company and their fulfillment

should be monitored by the system. This is especially useful to perform a contingency action if users unfortunately do not obey their duties. Similar to the dualism of permissions and prohibitions, duties have a counterpart as well: so-called *liberties*, which describe actions a user is free to do. Referring to modal or normative logic operators, there are four different circumstances: In certain situations a user is obliged to do something (1.), is obliged not to do something (2.), is free to do something (3.), or is free not to do something (4.).

In terms of modal logic, the first two concepts refer to a kind of necessity (1. *positive duty*; 2. *negative duty*), whereas the latter two refer to a kind of possibility (3. *positive liberty*; 4. *negative liberty*). Obviously, 1. and 4. as well as 2. and 3. may be in conflict with each other if they concern the same task. Thus, a conflict resolution policy is required. Duties and liberties are denoted as *normative rights* and hence, they are a subset of rights, just like permissions and prohibitions, which are denoted as *access rights* (see Figure 1).

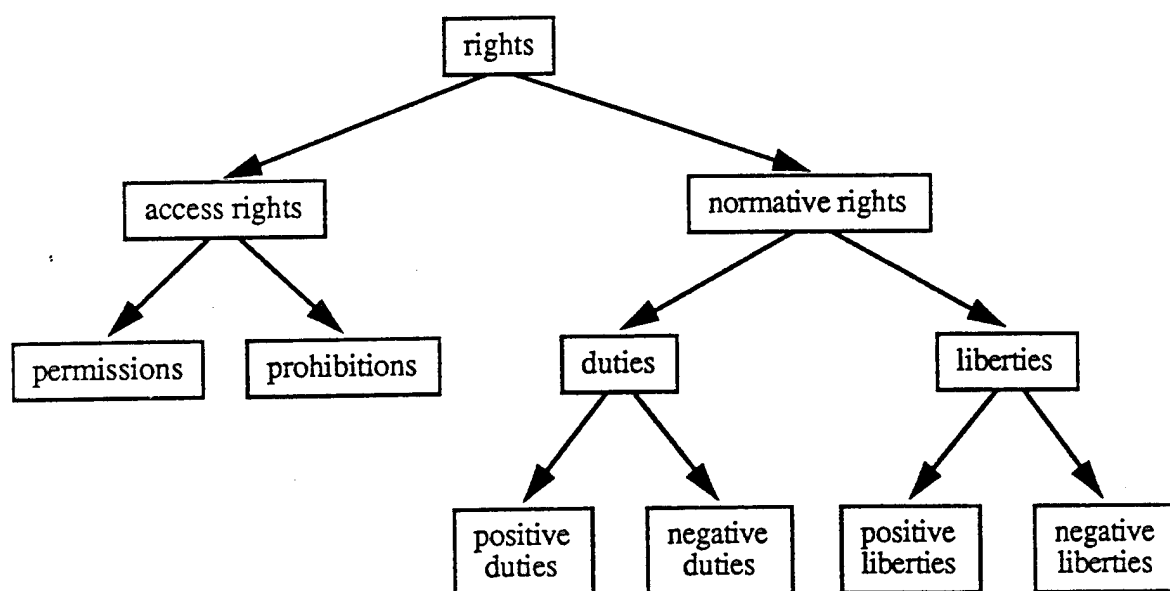


Figure 1. Categories of rights.

Obviously, there is a relationship between access rights and normative rights to be defined by the security policy. There are two different approaches possible, which are based on the following main principles: 1. A duty to do something implies the permission to do it, and 2. Duties and permissions are independent of each other, i.e. granting a duty requires a further grant of the corresponding permission, which is necessary to allow the fulfillment of that duty. This paper deals with both possibilities and shows the consequences of either choice.

In the following it is assumed that tasks are modelled as transactions. A task, of course, may sometimes require the execution of a set of different transactions, but from the access control point of view, it seems to be possible to consider this simplified as one nested transaction (cf. /Moss 81/). Nevertheless, it is clear, however, that a nested transaction is very different from simply a set of (sub)transactions. Concerning the access control policy, only transactions are considered as a unit of authorization. This simplification was made to concentrate on the relationships between access rights and normative rights. The ordinary authorization dealing with accesses to certain protection objects may exist as well. Thus, the starting point is similar to that one chosen for the Clark & Wilson model (/CIWi 87/), but note that the *duties* of their *separation of duties* - principle carry slightly different semantics.

The data model underlying the database is not important here, but with respect to the entire conception where these ideas are a part of, in future an object-oriented data model will be assumed. The active mechanisms are kept from HiPAC¹ (/Chak 89/, /Daya 88/ and /DaBC 88/) and SAMOS² (/GaGD 91/) and slightly modified to meet the requirements of this particular application.

The remainder of this paper is organized as follows. In Section 2 the model requirements are given, i.e. a very simple transaction model and the required active mechanisms are defined. This section is kept very short, because most of these ideas are already published. Then, the main ideas concerning the modelling process of the access behaviour of users are introduced: roles and tasks. In Section 4 the formal definition of rights is given (always with respect to transactions), followed by a comprehensive description of the relationships between access rights and normative rights (i.e. the mapping of normative rights onto access rights and triggers; Section 5) and a proposal of conflict resolution principles (Section 6). Section 7 deals with some ideas for decentralized authorization of normative rights and in Section 8 a database architecture to implement this concept is suggested. Finally, some related work is discussed.

2. Model requirements

Subsequently, no special data model is assumed. It is sufficient to have (nested) transactions and it is not important, whether these transactions are a sequence of data manipulation statements or a sequence of messages, activating certain methods assigned to objects, etc.

Let T be the set of transactions. A transaction is described as follows:

$$\forall t_k \in T: t_k = t_k (p_1 : \text{dom}(p_1), \dots, p_n : \text{dom}(p_n)),$$

i.e. transactions have a name (t_k stands for both the transaction and its name; hence the uniqueness of names is assumed) and a set of formal parameters p_i (either input or output) to be replaced by actual parameters, belonging to the corresponding domain, if this transaction is invoked.

Their internal structure is not important, because the nesting feature is only required to "bundle" complex tasks into a single authorization unit, nor are operational issues a point of consideration here (cf. /Moss 81/).

As mentioned in the introduction, the active mechanisms are taken from HiPAC and SAMOS. HiPAC uses the term *ECA-rule* for a trigger, giving a hint to the 3 main parts of them (Event, Condition, Action). An ECA-rule has the following structure:

- Identifier (an object-oriented environment was assumed)
- Event
- Condition
- Action
- Timing constraint
- Contingency plan
- Attributes

Events are divided into data manipulation events (either begin or end of operations for data manipulation), time events (absolute or relative time), external events (to be raised explicitly by an application) and transaction events (begin or end of transaction, abort, commit). Events may have formal parameters to specify a particular event from a class (or type) of events.

In HiPAC a condition is defined by a set of database queries, which is evaluated to be true, if all queries return a non-empty answer. Subsequently, it is assumed (similar to SAMOS) that a condition is any formula composed of predicates over the database state, which may include comparisons of parameters (of transactions and events) as well as query language expressions.

¹ High Performance Active Database Management System

² Swiss Active Mechanism-Based Object-Oriented Database System

An action is any executable module, written in any language, and it may contain database operations.

The remaining parts of such a rule (except the identifier) are optional. A timing constraint determines a kind of deadline. The contingency plan is executed instead of the specified action if an event was raised, the corresponding condition was evaluated to be true and the scheduler cannot complete the triggered action in accordance with the timing constraint. Like other objects in object-oriented systems, rules may have additional attributes.

The semantics is as follows: If an event is raised and the corresponding condition is evaluated to be true, the action is executed. Nested transactions serve as the execution model. Furthermore, different coupling modes exist between an occurrence of an event, the evaluation of the condition and the execution of the action, with respect to the triggering transaction: *immediately*, *deferred* (until the end of the triggering transaction), *decoupled and causal dependent* as well as *decoupled and causal independent*. A triggered transaction which is decoupled and causal dependent only commits if the triggering transaction commits, i.e. the scheduler must ensure this dependency (cf. /Chak 89/). This case is useful to deal with abstract events, for example. If a request is aborted, the triggered action also should become aborted. The other mode, however, is useful too. E.g. if a trigger monitors the intrusion into a security domain, a detected attempt requires a counteraction, even if the transaction causing that attempt was later aborted.

For this paper it is sufficient to assume that each imperative rule only consists of an event, a condition and an action (like "classical" triggers). Concerning the execution model, it is assumed that a transaction raising an event becomes interrupted, the conditions of associated rules are evaluated (within a set of parallel subtransactions; what corresponds to HiPAC's coupling mode *immediately*) and all triggered actions are executed within top-level transactions of their own, but dependent on the triggering transaction (what corresponds to HiPAC's coupling mode *decoupled and causal dependent*). Hence, the triggering transaction resumes if the evaluation of all conditions is finished. (Because of the "parallel" execution of the set of subtransactions evaluating the conditions, it would also be possible not to interrupt the triggering transaction.)

Furthermore, complex events are required. The following constructors were suggested:

- Disjunction of events: $(e_1 \mid e_2)$ (HiPAC, SAMOS)
(This allows for combining identical actions to be triggered from different events.)
- Sequence of events: $(e_1 ; e_2)$ (HiPAC, SAMOS)
(The rule is triggered if the second event follows the first one during the same transaction.)
- Closure of events: $(e_1^* ; e_2)$ (HiPAC)
(A discretionary repetition of the first event followed by the second event during the same transaction triggers the rule.)
- Conjunction of events: (e_1 , e_2) (SAMOS)
(This operator is equivalent to a sequence, which is independent of the order.)
- Negation of events: $\neg(e)$ (SAMOS)
(A rule is triggered if an event did not occur during a specified transaction or in a predefined time interval. Negative events must not occur at the beginning of a sequence of events to avoid problems of semantics.)

In the following, the constructors *conjunction*, *disjunction*, *sequence* and *negation* are used to build complex events. Similar to SAMOS, it is necessary to decouple complex events from the execution of a single transaction. SAMOS allows a specification of a time interval (a minimal as well as a maximal duration between the occurrence of atomic events) for a sequence and a conjunction of two events, which may be raised by different transactions. The default, however, remains to be a validity within the same transaction. For duties the opposite point of view is more appropriate, i.e. the default is that atomic events, which are part of a complex event, are raised by different transactions. Nevertheless, time boundaries for sequences or conjunctions of events remain to be useful too. Another enhancement is required for duties: Each atomic event within a complex event needs its own condition (in the simplest case "true") instead of the existence of only one condition for the entire complex event. In HiPAC and SAMOS the condition is evaluated if the entire complex event is raised, i.e. it is rather associated with the last

atomic event, which was raised. Unfortunately, it turns out that this simplification is inadequate for duties. Similar to the motivation in /Chak 89/ to separate events from conditions, it seems to be quite natural to associate each atomic event within a complex event with a separate (sub)condition.

A *situation* is a combination of an event and a condition. It is simply an expression "e:c" if *e* is an atomic event (either negated or not). Hence, "e:c" is an atomic situation. Complex situations are defined inductively. Let $s_1, s_2 \dots$ be situations. Then,

- $(s_1 \mid s_2)$
- $(s_1 ; s_2 [> t_{\max}])$
- $(s_1, s_2 [> t_{\max}])$

are situations, where in case of a sequence or a conjunction " t_{\max} " (optional) determines the maximum duration between the occurrence of the first and the occurrence of the last situation to keep the entire situation valid. The following statement is used to describe a trigger:

ON <situation> DO <action>³

3. Definition of roles and tasks

The modelling process of the access behaviour of users of a company (or any organization) can be based on the following two notions: *roles* and *tasks*.

In accordance with /JoGe 91/, a *role* describes the organizational, functional or social position of users within the UoD. Thus, roles are a kind of abstract users, within which concrete users can act, i.e. users can play a role. Typically, a user can play different roles. In most cases he will be associated with one organizational role (*personal manager, book-keeper a.s.o.*) and he can play different functional roles (e.g. *security administrator, head of a project group, trade-union official*, etc.). Organizational roles are similar to the more traditional notion of groups and they are appropriate to model the organizational structure of a company. In the following it is assumed that each user can play a set of predefined roles (either organizational or functional or social) and he must specify during the login procedure, which role(s) he actually wants to act in. During a session roles may become activated or deactivated by special commands (for example: "activate role <role_name>" and "deactivate role <role_name>").

Roles are also a unit of authorization, i.e. it is possible to grant and revoke rights to roles. Subsequently, permissions and prohibitions (*access rights*) are defined as usual for discretionary access control policies (cf. Section 4.). Users, playing a role, inherit the corresponding rights. Besides that, rights can be granted and revoked to concrete users too, especially to deal with exceptions, i.e. to enhance or to override the rights inherited from certain roles. (The term *subject* refers to both roles and users.) If a user can act in more than one role at the same time, the corresponding rights are combined. (See Section 6 for conflict resolution.) Although some role combinations are not critical with respect to the pile up of rights (e.g. *operator* and *head of a project*), others would give the opportunity to collect rights, which should never become applicable at the same time (e.g. *doctor* and *patient*).

It must also be taken into consideration that a user, acting in at least two roles at the same time, can establish an unintended information flow from one role to the other. Whereas a user may be allowed to read a protection object *A* (inheriting the necessary permission from role *X*) and to read as well as to write into a protection object *B* (inheriting the corresponding permission from role *Y*), it could be possible that other users who can only act in role *B*, must not know the content of protection object *A*. If the former user can act in both roles at the same time, it is very easy to write the content of *A* into *B* (if the corresponding domains match each other), thereby establishing an unintended information flow. Preventing an activation of both roles (*X* and *Y*) at the same time would "thwart" this possibility. (Of course, that user may activate role *X*, read *A*, remember its content or write it down, deactivate *A*, activate *B* and reenter

3 Possibly, it is useful to retain a global condition, i.e. to define a trigger in the following manner:

ON <situation> IF <condition> DO <action>

the content of A into B . But this is a very cumbersome way, which is almost never to prevent if discretionary policies are applied.) A similar situation can happen because of an execution of several transactions.

So, the security administrator has to establish an **activation conflict relation** to determine which roles must not be activated at the same time. The access control system must enforce these constraints. The specification could be done by a table with boolean entries, the rows and columns being labelled by all defined roles. If the entry of row X and column Y is "false", the system refuses an activation of these roles at the same time. Otherwise, a concurrent activation is possible if the user is a member of both roles. Obviously, this relation is irreflexive and symmetric.

Note that this is different from the *separation of duties-principle* of the Clark&Wilson-policy (CIWi 87/). To have a means of ensuring that principle another declaration is necessary: an **association conflict relation**. The assignment of users to roles may change in time and the security administrator may forget which roles a user can already play. So he can unintentionally make a dangerous mistake (e.g. assign a user to the role *auditor* who is already assigned to the role *book-keeper*). Hence, it is better to offer system support. The association conflict relation prescribes which roles have to be strictly separated, i.e. for which it is impossible to associate the same user with both roles at the same time.

Roles may be in a relationship of sub- and superordination. This is especially useful for organizational roles to model the structure of a company. The binary relationship " \leq " between roles describes that situation. Let R be the set of roles (either functional, organizational or social). Then, (R, \leq) forms a poset (partial ordered set, i.e. the relation is reflexive, antisymmetric and transitive). A role r_i is subordinated to r_k if " $r_i \leq r_k$ " holds. Neither is required that each role (except a "super-role") has at least one superior role, nor is required that a role may have at most one superior role. These relationships are used to inherit rights.

Tasks are associated with subjects (mainly with roles) and describe their *responsibility domain*. Tasks are bundled to (nested) transactions during the modelling process, by uniquely mapping each task (as a logically related unit of actions) to one transaction. Furthermore, tasks are the source of deriving the rights necessary for a role to fulfil its tasks, i.e. they are the conceptual basis for the authorization process to assign the appropriate rights to subjects.

4. Definition of rights

Subjects (either roles or users) are the unit of authorization. An authorization of users allows for exception handling. (Sometimes it is required that certain users, who are assigned to a role, have more or even fewer rights than other users of that role.)

The relationship between a user of a computer system and the process acting on his behalf is established by an identification and authentication procedure. Note that always only one process is associated with a user, but the effective rights of that process may change, depending on activated roles and on decentralized and dynamic authorizations. Let S be the set of subjects and U the set of users. It obviously holds: $S = R \cup U$ and $R \cap U = \emptyset$

Furthermore, a discretionary policy with the closed world assumption is assumed. Note that a prohibition is stronger than a non-existing permission.

Definition of permissions:

A permission is a five-tuple: (s_1, ta, p, f, s_2)
 s_1 and s_2 are subjects, ta is a transaction, p is a predicate defined by the formal parameters of ta (either input or output) and f is a grant flag. The predicate allows to restrict the execution of ta , e.g. depending on the protection objects to be accessed.

The semantics is that subject s_1 is permitted to execute transaction ta with actual parameters fulfilling p . Furthermore, this permission was granted by s_2 . (The grantor is always a user and

not a role!) The subject is allowed to grant this permission again to another subject if and only if the grant flag is *true*.

The "classical" relationship of discretionary policies established between a subject, a protection object and an action is somewhat hidden, because the protection objects are only accessed by transactions. Let PE denote the set of permissions.

The authorization of permissions could be realized by means of the following commands (clauses enclosed in square brackets are optional):

GRANT PERM ta WITH p TO s_1 [WITH GRANT OPTION]
 REVOKE PERM ta WITH p FROM s_1

The same grantor can give different permissions to the same subject regarding the same transaction. Hence, the predicate serves as an identifier to revoke the right permission. It is not a point of discussion here, whether this is a good idea or whether another scheme would be more appropriate, e.g. the association of explicit identifiers to permissions.

Definition of prohibitions:

Similar to permissions a prohibition is a four-tuple: (s_1, ta, p, s_2)
 s_1 and s_2 are subjects, ta is a transaction and p is a predicate defined by the formal parameters of that transaction (either input or output).

The semantics is that subject s_1 is forbidden to execute transaction ta with actual parameters fulfilling p . This prohibition was granted by user s_2 . A grant flag does not make any sense here, because it is semantically meaningless that somebody can transfer his prohibitions (which he usually should not know!) to another subject. Let PR denote the set of prohibitions.

The authorization of prohibitions could be realized by means of the following commands:

GRANT PROH ta WITH p TO s_1
 REVOKE PROH ta WITH p FROM s_1

Definition of duties:

A duty is a nine-tuple: $(s_1, si_1, k, ta_1, p, si_2, ta_2, f, s_2)$
 s_1 and s_2 are subjects, si_1 and si_2 are situations, k is a flag determining the kind of duty ("1" - positive duty; "0" - negative duty), ta_1 and ta_2 are transactions, p is a predicate defined by the formal parameters of ta_1 , and f is a delegation flag.

The semantics is as follows. Positive duty ($k=1$): If situation si_1 occurs (i.e. the associated (possibly complex) event was raised and the condition(s) was (were) evaluated to be true), subject s_1 is obliged to execute transaction ta_1 with parameters fulfilling p . Thus, ta_1 is a transaction which requires interaction. Otherwise, an automatic execution would be possible, as usual for triggers. Since it is possible that s_1 does not obey his duty, the system monitors the fulfillment for a specified time. The particular time interval, in which an execution is required, is bounded by a second situation si_2 (in the simplest case it is only a relative time event given with respect to the triggering situation, and a "true" condition). The system schedules transaction ta_2 automatically (with an appropriate replacement of the formal parameters) if situation si_2 occurs and the subject has not yet executed transaction ta_1 . So, the second transaction is a little bit similar to the "contingency plans" of HiPAC, but its motivation and semantics is very different. The flag f determines whether this duty may be delegated to another subject ($f=true$) or not ($f=false$; cf. Section 7). Furthermore, this duty was granted by user s_2 .

Negative duties ($k=0$) are very different: If situation si_1 occurs, subject s_1 is obliged not to execute transaction ta_1 with parameters fulfilling p , until situation si_2 occurs. No "contingency plan" is necessary, because the access control system can enforce negative duties. Furthermore, it is not meaningful for negative duties to have a delegation flag. Hence, the following consistency constraint holds (Let D denote the set of duties.):

$$\forall d \in D \ (d = (s_1, si_1, k, ta_1, p, si_2, ta_2, f, s_2)) : k = 0 \implies ta_2 = \emptyset \wedge f = false$$

Duties can be written in the following manner, which is easier to understand:

- positive duties: ($s_1, si_1, 1, ta_1, p, si_2, ta_2, true, s_2$)

s_1 : ON si_1 DO ta_1 WITH p UNTIL si_2
 ELSE ta_2

WITH GRANT OPTION

(If f is false, the last clause is dropped.)

- negative duties: ($s_1, si_1, 0, ta_1, p, si_2, \emptyset, false, s_2$)

s_1 : ON si_1 DO NOT ta_1 WITH p UNTIL si_2

With respect to /JoGe 91/, the following changes were made:

- ☞ Situations combine events and conditions allowing a more consistent treatment of complex events.
- ☞ The second event was completed by a condition and becomes a situation too.
- ☞ A delegation flag was introduced (cf. Section 7).

The authorization of duties could be done by means of the following commands:

GRANT DUTY ON si_1 DO [NOT] ta_1 WITH p UNTIL si_2
 [ELSE ta_2] TO s_1 [WITH GRANT OPTION]
 REVOKE DUTY ON si_1 DO [NOT] ta_1 WITH p UNTIL si_2
 [ELSE ta_2] FROM s_1

Definition of liberties:

A liberty is a seven-tuple: ($s_1, si_1, k, ta, p, si_2, s_2$)

s_1 and s_2 are subjects, si_1 and si_2 are situations, k is a flag determining the kind of liberty ("1" - positive liberty; "0" - negative liberty), ta is a transaction and p is a predicate defined by the formal parameters of ta .

The semantics is as follows. Positive liberty ($k=1$): If situation si_1 occurs, subject s_1 is free to execute transaction ta with parameter fulfilling p . The intention is that liberties are used to restrict the validity of duties (either to shelter a user from an arbitrary suffering of duties granted by another user, e.g. his boss, or to have a means of overriding certain duties in case of a parallel activation of duties which are in conflict with each other.) Hence, liberties are the counterpart of duties. Analogously to duties, liberties are only valid for a specified time, which is bounded to the triggering situation si_1 and a second situation si_2 (in the simplest case also a relative time event with respect to the triggering situation and a "true" condition.) A grant flag seems not to be meaningful, but this may depend on the security policy chosen by the customer. Furthermore, this liberty was granted by user s_2 .

Negative liberties ($k=0$) are similar: If situation si_1 occurs, subject s_1 is free not to execute transaction ta with parameters fulfilling p until situation si_2 occurs. Let L denote the set of liberties.

Liberties can be written in the following manner, which is easier to understand:

- positive liberties: ($s_1, si_1, 1, ta, p, si_2, s_2$)

s_1 : ON si_1 BE FREE TO DO ta WITH p UNTIL si_2

- negative liberties: ($s_1, si_1, 0, ta, p, si_2, s_2$)

s_1 : ON si_1 BE FREE NOT TO DO ta WITH p UNTIL si_2

With respect to /JoGe 91/, liberties are used in a completely different way. In this older publication, liberties are not used as a counterpart of duties, but as a special kind of permissions to describe communication relationships between subjects. But it seems to be more convincing to use liberties in the way described in this paper.⁴

⁴ The idea to do so stems from Dr. Brüggemann from Hildesheim University.

The authorization of liberties could be done by means of the following commands:

```
GRANT LIB ON  $s_1$  BE FREE [NOT] TO DO  $ta_1$  WITH  $p$  UNTIL  $s_2$  TO  $s_1$ 
REVOKE LIB ON  $s_1$  BE FREE [NOT] TO DO  $ta_1$  WITH  $p$  UNTIL  $s_2$  FROM  $s_1$ 
```

Example:

The example is taken from a (simplified) bank application. Subsequently, some normative rights are described which may concern a bank-teller, who pays out money (surprisingly) and may decide about overdraft provisions, if the borrowed amount does not exceed a specified limit. Assume, a customer is admitted to overdraw his account at most by 10.000 DM and in case of an overdraft between 5.000 and 10.000 DM it is free to the bank-teller to accept this or not. (Hence, he is also responsible for his decision!)

Let "request(c, ac, am)" be the event that a customer c wants to debit an amount am from account ac . The corresponding transaction is "debit(c, ac, am)", to be executed by the bank-teller to serve this request. The operation "balance(ac)" determines the current balance of an account ac and the operation "time(e)" returns the time when event e was raised.

Now, a bank-teller has the positive duty to pay out the requested money if the new balance of the account does not fall below the limit of -5.000 DM (assuming that a bank-teller always has the permission to do so):

```
bank-teller: ON ( request( $c, ac, am$ ) : (balance( $ac$ ) -  $am$  >= -5 000) )
DO debit( $c1, ac1, am1$ ) WITH ( ( $c1=c$ )  $\wedge$  ( $ac1=ac$ )  $\wedge$  ( $am1=am$ ) )
UNTIL (time(request( $c, ac, am$ )) + 5 min : true)
ELSE "order a coffee for the customer"
```

The ELSE-clause (which is obviously not formalized here) may be changed corresponding to the style of the bank. If a duty is granted to a role (e.g. *bank-teller*), it is sufficient that one user who currently plays that role obeys it.

He has the negative duty not to pay out money if the new balance would fall below the limit of -10.000 DM. This overwrites the "global" pay-out permission of a bank-teller, because negative duties imply temporal prohibitions (cf. Section 5), which take priority over permissions:

```
bank-teller: ON ( request( $c, ac, am$ ) : (balance( $ac$ ) -  $am$  < -10 000) )
DO NOT debit( $c1, ac1, am1$ ) WITH ( ( $c1=c$ )  $\wedge$  ( $ac1=ac$ )  $\wedge$  ( $am1=am$ ) )
UNTIL (credit( $c2, ac2, am2$ ) :
(ac= $ac2$ )  $\wedge$  (balance( $ac$ ) -  $am$  +  $am2$  >= -10 000)  $\wedge$ 
request( $c3, ac3, am3$ ) : ( $c=c3$ )  $\wedge$  ( $ac=ac3$ ) )
```

Hence, it is impossible for the bank-teller to debit this account until somebody credits a sufficient amount to this account or the customer changes his request. In the latter case it is not necessary to check the overdraft-condition. Of course, this new request invalidates the activated negative duty, but a new one is triggered if the new request does not fulfil the condition!

A bank-teller has the positive liberty to pay out money if the new balance of an account is between -5.000 and -10.000 DM, which invalidates any negative duty not to do so. On the other hand, he is also responsible for the consequences!

```
bank-teller: ON ( request( $c, ac, am$ ) : (balance( $ac$ ) -  $am$  >= -10 000) )
BE FREE TO DO debit( $c1, ac1, am1$ )
WITH ( ( $c1=c$ )  $\wedge$  ( $ac1=ac$ )  $\wedge$  ( $am1=am$ ) )
UNTIL request( $c2, ac2, am2$ ) : ( $ac=ac2$ ) )
```


Three different decisions concerning positive duties are imaginable:

- ☛ Granting a positive duty always implies granting the corresponding permission too, whether the grantor is allowed to grant this permission explicitly or not (cf. Section 7). This coincides with policy (1).
- ☛ Granting a positive duty never implies granting the corresponding permission. This coincides with policy (2). Therefore, two subjects must collaborate to make a user suffering a duty if the grantor of a duty is not able to grant the required permission too. This situation can occur in case of certain *separation of duties* requirements in accordance with the chosen security policy (*duty* in the Clark&Wilson-sense).
- ☛ Granting a positive duty implies granting the corresponding permission if and only if the grantor is allowed to grant this permission too (cf. Section 7).

Independent of this choice, the authorization system always generates the corresponding trigger monitoring the fulfillment of a duty. Hence, at least the "contingency plan" is scheduled if the subject suffering a duty has not the required permission to obey it.

By the way, the condition of the second situation within the sequence of events of this trigger is necessary because it is not sufficient that subject s_i simply executes the transaction. The parameters of the execution must fulfil the required predicate to ensure that really the associated task was done and not something else.

Analogously, the same choices are possible for negative duties. However, negative duties may become a redundant concept, depending on the security policy chosen. They are meaningless if negative duties do not imply a prohibition and liberties always take priority over duties (as well as prohibitions over permissions). The access control system can always ignore them in such a case.

If positive liberties are mapped onto permissions, they get another semantics than introduced in Section 4. However, it is not contradictory to the assumptions made up by now to use liberties to override duties and to deduce a corresponding permission. Then, however, the authorization system must ensure that these implied permissions cannot be overridden by prohibitions. This causes some problems, because, in general, prohibitions take priority over permissions (cf. Section 6). This problem is solvable if explicit priorities are assigned to rights (cf. Brügg 91/) and the system ensures that each permission implied by a positive liberty always gets a higher priority than any prohibition. A negative liberty is only meaningful to override a positive duty.

Furthermore, the implied access rights have to be valid only within the time interval bounded by the specified situations, whether these access rights are implied or not. Such a temporal authorization can be based on active mechanisms too. Subsequently, this is exemplified for positive duties: Two triggers are required. The first grants the permission if the triggering situation of the corresponding duty occurs, and the second revokes that permission if the duty was obeyed or if the second situation occurs. Depending on the chosen policy, the authorization system can generate these triggers automatically. (Then, the grantor of the duty becomes the grantor of the permission too.)

- ON si_1 DO GRANT PERM ta_1 WITH p TO s_1
- ON ($si_1 ; (EOT(s_1, ta_1, \langle par_list \rangle) : p(\langle par_list \rangle) \mid si_2))$
DO REVOKE PERM ta_1 WITH p FROM s_1

Unfortunately, this is a little bit cheated if an explicit authorization of the access rights corresponding to normative rights is required, because an authorization for defining triggers is not a topic of this paper. (For the sake of simplicity it was assumed that triggers are only definable by the security administrator or implicitly generated by the access control system.) But the reader may imagine that this problem is solvable.

6. Conflict resolution

In this section, a particular security policy was chosen to exemplify the resolution of conflicts. There are many other possibilities imaginable (some of them are briefly mentioned at the end of this section), but it should be easy to conclude the required modifications. The following rules are taken into consideration (for this example!):

- Users possess rights, which are either explicitly granted to them or inherited from activated roles (in accordance with the activation conflict relation).
- Roles inherit permissions from their subordinated roles.
- Roles inherit prohibitions from their superior roles.
- Duties and liberties are not inherited. (At least duties should never be inherited, since they are typical of a subject, whereas the inheritance of liberties may be a matter of taste.)
- Positive duties do not imply any permission. The required permissions have to be granted explicitly by temporal authorizations.
- Negative duties imply the corresponding temporal prohibitions.

Hence, there are two basic conflicts to be considered, concerning rights defined for the same subject and the same transaction:

- ☞ contradictions between permissions and prohibitions and
- ☞ contradictions between duties and liberties
(more precisely, between positive duties and negative liberties and between negative duties and positive liberties)

With respect to the relationship between permissions and prohibitions, it is assumed that prohibitions always take priority over permissions. Nothing else is required to check the validity of an execution request of a transaction, since positive duties do not imply permissions and negative duties imply prohibitions. Let u be a user who wants to execute transaction ta with parameters $\langle par_list \rangle$. Furthermore, let R' be the set of roles activated by u . Hence, the access is allowed if the parameters fulfil a predicate of any applicable permission and if they do not fulfil a predicate of any applicable prohibition:

$$\begin{aligned}
 u \text{ is allowed to execute } ta \text{ with } \langle par_list \rangle &\iff \\
 \exists (s_1, ta, p, f, s_2) \in PE: &p(\langle par_list \rangle) \wedge \\
 &(s_1 = u \vee (s_1 = r \wedge (r \in R' \vee (\exists r' \in R': r \leq r'))))) \\
 \wedge \nexists (s_1, ta, p, s_2) \in PR: &p(\langle par_list \rangle) \wedge \\
 &(s_1 = u \vee (s_1 = r \wedge (r \in R' \vee (\exists r' \in R': r \geq r')))))
 \end{aligned}$$

Note that this is a very restrictive policy. If a user has activated two roles, prohibitions valid for one role may override permissions valid for the other. Thus, the strange situation is imaginable, where a user, who activates an additional role, only suffers a loss. However, even such a situation could be intended. Furthermore, this policy prevents undesirable information flows as described in Section 3. There are other possibilities to solve such conflicts (cf. /JoGe 91/). A promising one is to assign explicit priorities to rights, e.g. by means of integers and choose the right with the highest priority to check a request. Conflicts arising at the highest level, may either be rejected by the authorization system (/Brüg 91/) or be resolved by the scheme described above.

The relationship between duties and liberties is the more interesting case for this paper. In order to meet the intention of liberties, i.e. to give users a shelter from an arbitrary suffering of duties or to override contradictory duties, liberties take priority over duties. Due to the mighty specification facilities for situations, it is not possible to check for conflicts at authorization time. Normative rights are valid for a period bounded by two situations. If only (absolute) time events trigger normative rights and the validity scope is given by another absolute time event or a relative time event (relative to the triggering time event), conflicts could be detected in advance. Generally, however, this is impossible. Therefore, a monitor is required which reacts on the particular situation and controls the temporal authorization. (Subsequently, it is assumed

that always the same transaction and the same subject are concerned. Otherwise, there would not be any conflict.)

(1) A positive duty is triggered:

In accordance with the chosen policy, the monitor has to check if a negative liberty is valid (or even more than one liberty). (*Valid* means that this liberty was triggered and still not annulled by the second situation.) If not, there is no problem.

However, if a liberty is currently valid, nevertheless, the duty is also activated, because it is free to the user to obey his duty or not. Hence, if the user agrees to do the job, he needs the corresponding permission (explicitly granted) and so, the temporal authorization must take place and the trigger is activated too (with a high probability to be fired).

Hence, the question may arise why to make use of negative liberties? Are they redundant in case that such a policy was chosen? The answer depends on the respective application. Generally, however, it is useful that the system gives a hint to the grantor of this duty (e.g. by mail) that the grantee is free not to obey the duty (until the second situation of the liberty occurs). So, the grantor has a chance to react to this circumstance, e.g. to ask the grantee if he agrees to do this job, or to do it by himself. This seems to be better for the operation of a company than the later (possibly too late) recognition that the grantee has not done the job, due to a valid negative liberty.

(2) A negative duty is triggered:

This case is more complicated than (1). If no positive liberty is valid, there is no problem and the corresponding prohibition is temporally granted.

However, if such a liberty is valid, it is free to the user to execute the transaction. Hence, it is impossible simply to grant the corresponding prohibition. Since positive liberties override negative duties, it is necessary to conjoint the predicate of the negative duty with the negation of the predicate of the liberty (if more than one liberty is activated, the corresponding predicates must be disjoint first). Hence, the temporal prohibition has to be granted with the predicate: $"p_{\text{negative duty}} \wedge \neg (p_{\text{positive liberty}})"$

Similar to (1), a message to the grantor of the (partially or completely) overridden duty is recommendable.

(3) A positive liberty is triggered:

If no negative duty is valid, no reaction of the monitor is required. Otherwise, the temporally granted prohibition corresponding to that duty, must be modified in accordance with (2), i.e. the implied prohibition has to be partially revoked.

(4) A positive liberty is annulled:

If a negative duty is valid, a modified temporally granted prohibition exists (in accordance with (2)). This modification must be undone, i.e. the predicate of the corresponding prohibition must be restored. (If there remain some other valid positive liberties, only the clause of the deactivated liberty within the disjunction " $p_{\text{positive liberty}}$ " has to be removed.)

(5) A negative liberty is triggered:

If there is a valid positive duty, i.e. a duty which was not yet obeyed by the corresponding subject (otherwise it would be annulled in the meantime), analogously to (1), a message to the grantor is recommendable that from now (until the terminating situation occurs) it is free to the grantee not to obey this duty. (bad luck for the grantor)

(6) A negative liberty is annulled:

If there is a valid positive duty, a message to the grantor is recommendable too, to indicate that the time is expired, where it was free for the grantee not to obey this duty. Hence, it can be expected that the job will be done. (bad luck for the grantee)

There are other policies imaginable for the relationship between duties and liberties. It is also possible to override the duty in case of a conflict between a positive duty and a negative liberty in such a way that only the temporal permission is granted, but the monitoring trigger is not activated. The problem is that the predicates of the liberty and the duty may be different. Especially, they may overlap with respect to the sets of actual parameters fulfilling them. In such a case, the duty could be activated with the predicate of the duty conjoined with the negation of the predicate of the liberty (whereas the corresponding permission remains to be the same). Unfortunately, in general it is not decidable whether the resulting predicate is a contradiction (and therefore, it is meaningless to activate this duty) or not. So, this simplified policy was chosen. Further investigations are required to examine whether the predicates are usually simple comparisons of formal and actual parameters, i.e. whether only "simple predicates" occur (cf. the examples given in Section 4!). Then, this problem would be solvable!

The association of explicit priorities to normative rights is another possibility (analogously to access rights). In case of a conflict, the right with the highest priority overrides the others (whether duty or liberty). Remaining conflicts at the highest level can be treated in the way described above.

7. Decentralized authorization of normative rights

Although decentralized authorization of access rights does not concern the main subject of this paper, some remarks about it seem to be reasonable.

Every new or changed transaction, to be executed within the system, must be evaluated first (/CIWi 87/). This is done by users belonging to a role *evaluator*. (For simplicity, it is assumed that there exists only one such role. However, it is no problem to designate several *evaluator* roles, corresponding to different sets of subjects who are developing transactions. It is also possible that each role receives an evaluate-privilege with respect to its subordinated roles, or that, corresponding to an ownership-paradigm, the user who has written a transaction, may evaluate it too.)

An authorization is only possible if the transaction has already been evaluated. Every user who has evaluated a transaction, receives automatically a prohibition to execute this transaction (with the predicate "true"), to ensure the *separation of duties*-principle of the Clark&Wilson-policy. (How depressing is the live of a censor!) This decision is not meaningful, however, if another policy was chosen, e.g. if the evaluation is done by the owner or by a superior role!

Furthermore, an *authorizer* role is responsible to grant and revoke appropriate access rights. (Similar to *evaluators*, it is possible to designate several *authorizer* roles or to give an authorization-privilege to the owner.) If a permission was granted with grant option, the grantee is free to grant this permission again, either with grant option or not. Then, the restriction predicate of the grantor is inherited to the grantee and if the grantor requires a further restriction, it is conjoined with his own predicate. *Authorizers* may restrict the possible broadcasting of permissions, either by omitting the grant option or by virtue of preventive prohibitions. Hence, if such an unlucky subject (who "owns" a preventive prohibition) receives a decentralized granted permission, it is overridden by the prohibition existing already.

Each user can revoke rights, which he has granted by himself. (This is the main reason to introduce a grantor attribute for rights.) *Authorizers* can revoke every existing access right. In case of revoking a permission, which was transitively granted to other subjects, these permissions are revoked too. (See /JoGe 91/ for a restricted revoke.)

Furthermore, it is possible to relax the *separation of duties*-principle stated above, by including *evaluator* and *authorizer* into the association conflict relation (instead of granting a prohibition automatically). Hence, nobody can evaluate a transaction and grant a permission to execute it to himself. However, similar to a "four-eyes-principle" it is possible that an evaluator receives a permission from another user.

Of course, this is only a very simple scheme, but normative rights are emphasized in this paper.

A user may grant duties (either positive or negative) to himself and to roles which are subordinated to at least one of those roles, the user has currently activated. Furthermore, he is allowed to grant duties to particular users if they belong to at least one role for which a superior role exists which is currently activated by the grantor. Let R_1 be the set of roles activated by a user u and R_2 the set of roles subordinated to any role of R_1 . Hence:

$$\forall r_1 \in R_2 \exists r_k \in R_1 : r_1 \leq r_k$$

$$u \text{ is allowed to grant a duty to a subject } s \iff s = u \vee s \in R_2 \vee \exists r_k \in R_2 : s \in r_k$$

If a user receives a duty with grant option (in case of a positive duty), he is allowed to delegate this duty to subordinated roles or to users belonging to a subordinated role. In case of such a delegation, the grantor gets rid of his duty, i.e. from now on, only the grantee is obliged to obey this duty. Although the user is also able to perform this grant to subordinates if he has not the grant option, this makes a difference, because in the latter case, he retains the duty and has not gained any advantage.

No user can grant a liberty to himself. Similar to duties, a user is allowed to grant liberties to subordinates. However, liberties were mainly introduced to shelter users from suffering arbitrary duties (i.e. to shelter them from moods of their bosses). For this purpose, a particular role *liberator* is introduced (similar to *evaluator* and *authorizer*, it is possible to define several liberators for different sets of subjects). With the notation introduced above holds:

$$u \text{ is allowed to grant a liberty to a subject } s \iff s \in R_2 \vee (\exists r_k \in R_2 : s \in r_k) \vee \\ u \in \text{liberator}$$

There are situations possible, where a user grants a duty to one of his subordinates who already possesses a conflicting liberty (or vice versa) which was granted by another user. It is hard to give a meaningful criterion to solve this kind of contradictions, because it is even possible that both competing grantors are not in a relationship of sub- and superordination to each other. So it seems to be more appropriate to solve this problem "outside the computer system". (In the Middle Ages the grantor of the duty would have had the choice of weapons, because his right was overridden.) Such an "outside solution" is supported, because the grantor of an overridden duty receives a message from the monitor that this duty is possibly not obeyed. Until an agreement, the system decides in favour of the concerned user, i.e. the liberty takes priority.

8. Enhancing HiPAC-architecture to treat duties

An architecture for an active database system (HiPAC) was proposed in /Chak 89/. Subsequently, this architecture is taken (slightly changed) and enhanced by the functionality required to deal with duties (cf. Figure 2). Certainly, there are other possibilities.

The original HiPAC-architecture was enhanced by the access control system, the monitor for normative rights (as part of the rule manager) and an authorization interface. The object manager was renamed to data manager, because no particular data model was assumed here.

First, the "pure" active components are briefly described in accordance with /Chak 89/. The rule manager is the heart of the system, which controls the firing of imperative rules. In case that an event was signalled, it selects the corresponding triggers, initiates the required condition evaluations, monitors complex events and calls the transaction manager (either to execute triggered actions or to serve requests for "ordinary" transactions). The management of rules is done by the data manager. According to rule definition and manipulation operations, the rule manager changes the affected rules and programs the event detectors as well as the condition evaluator. Furthermore, it keeps track of older database states required to evaluate conditions. The latter is necessary if the coupling mode between an event and the condition is not *immediately*. Therefore, this functionality is not required for duties. Apart from this, another coupling

mode would cause some problems with respect to complex situations which were introduced in this paper.

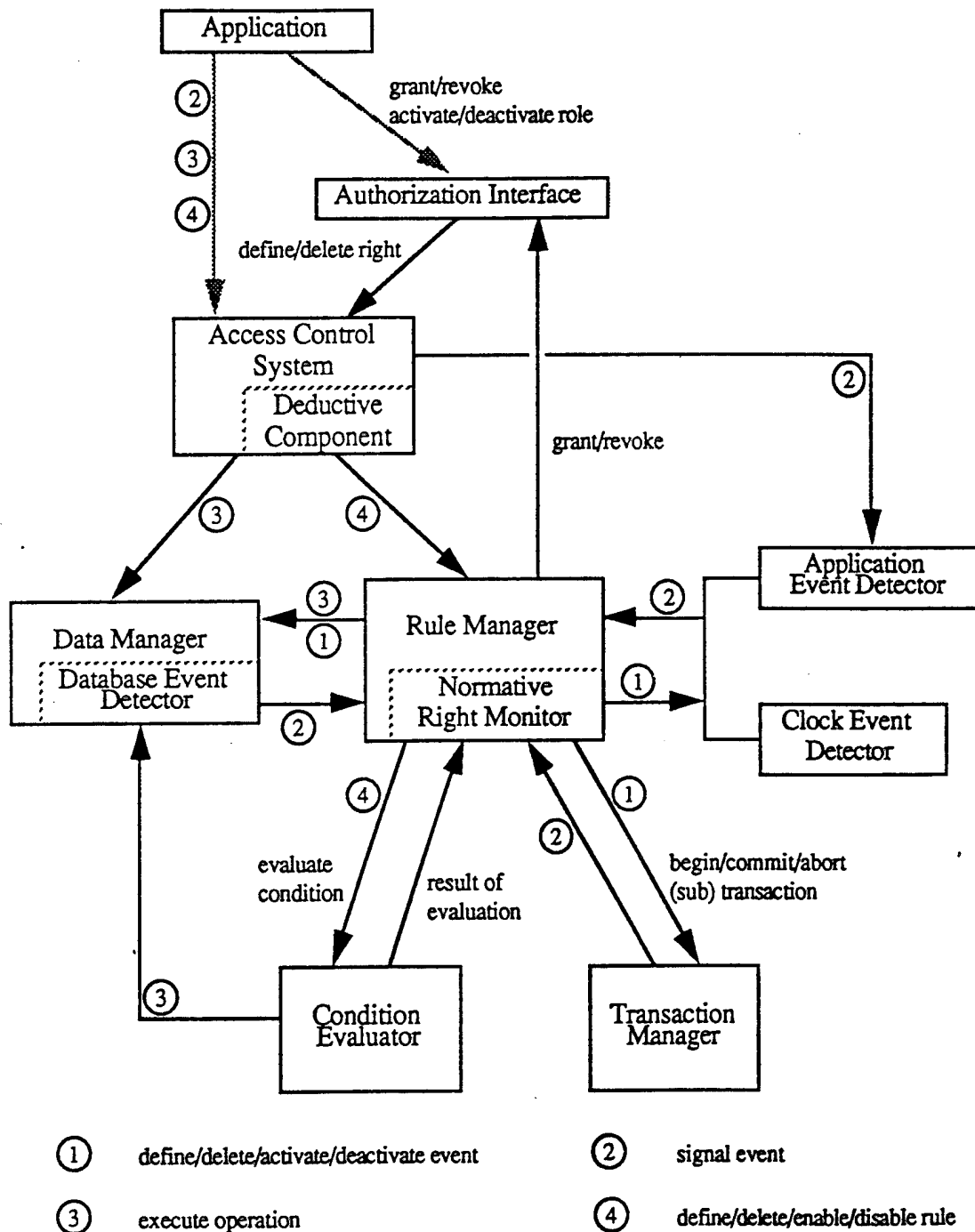


Figure 2. Enhanced HiPAC+ architecture.

Event detectors are multiple instances of the same generic component, corresponding to the type of events to be detected. The condition evaluator checks whether a particular condition is

true or not. The transaction manager as well as the data manager offer the same functionality as usual for database systems, enhanced by an event detector for transaction or database events, respectively. In addition, the transaction manager supports the cause and effect relationship between transactions. (See /DaBC 88/, /Daya 88/ and /Chak 89/ for a more comprehensive description.)

The access control system has to check any request to execute a transaction. The defined rights and the conflict relations are stored as a part of the database (more precisely, as a part of the data dictionary). Descriptive rules to deduce implicit rights (remember that roles inherit permissions from subordinated roles and prohibitions from superior roles) and to solve conflicts between permissions and prohibitions are used by the deductive component to determine the valid access rights, i.e. the rights that are not overridden by another right, depending on the chosen policy for conflict resolution. Note that the functionality of a first order calculus is not required, but both sets of rules must be separated. The process to deduce valid access rights is subdivided into two phases:

1. Inheritance of access rights
2. Conflict resolution

Each phase makes use of its own rule base. (A one-phase process would be more complicated.)

If a request was made, the access control system checks whether an applicable permission is deducible, i.e. whether a permission and no prohibition is valid with respect to the actual parameters of that request. In doing so, all currently activated roles have to be taken into consideration. If the request is allowed, it is passed to the rule manager who calls the transaction manager. (Possibly a direct call from the access control system to the transaction manager is more appropriate, but this depends on the "basic" architecture of an active database system to be enhanced with duties.) If the predicates of the corresponding access rights are also defined by output parameters, the transaction is executed and afterwards the access control system checks whether the execution was allowed and the output can be given to the user or whether the request must be rejected. In the latter case, the execution is undone and an error message is returned to the user. (This was not included in Figure 2, which is probably, even with this simplification, hard enough to understand.) Hence, the access control system acts like a reference monitor.

In case of a dynamic authorization, the access control system checks whether this authorization is allowed and changes the set of rights which are stored in the database by an appropriate call to the data manager. If a positive duty was granted, the access control system creates the corresponding trigger by virtue of the *define rule* operation, which is provided by the interface of the rule manager. Similar to this, if a duty was revoked, the corresponding trigger is deleted too.

The monitor for normative rights becomes a part of the rule manager. This is quite natural, because it has to supervise situations in the same way as the rule manager does. If a normative right is triggered, the monitor makes use of the authorization interface to (temporarily) grant and revoke the appropriate access rights, i.e. to change the authorization state in accordance with the policy chosen for conflict resolution (cf. Section 7). Hence, the monitor must know which policy was chosen for the relationship between access rights and normative rights (i.e. if there are some rules to imply access rights by normative rights, they are a part of the monitor) as well as to solve conflicts arising between normative rights. Thus, the monitor comprises a simple deductive component too.

9. Related work

A transaction-based authorization scheme was also proposed by Clark and Wilson in /CIWi 87/. Their approach was mainly driven by the aim to insure integrity of data by means of *well-formed transactions*. These transactions are the one and only way to manipulate data. The assignment of transactions to protection objects (Clark and Wilson have denoted them *constrained data items*) as well as to users being allowed to execute them was monitored to ensure integrity and secrecy. Of course, the evaluation of transactions to be *well-formed* by the security admin-

istrator is rather discretionary. One of the main principles of their policy was that someone who evaluates a transaction must not be allowed to execute it (*separation of duties*). This principle can be implemented with the approach proposed in this paper by virtue of a clean association of tasks with roles and an integration of conflicting roles (e.g. *evaluator* and "*executor*") into the association conflict relation. Furthermore, Clark and Wilson divide the procedures to manipulate data into two classes: *integrity verification procedures* and *transformation procedures*. Protection objects may only be manipulated by transformation procedures. The policy is described by a set of *enforcement rules* and a set of *certification rules*. The most important ones are that all transformation procedures must be certified (evaluated) and that a list describes which users may manipulate which protection objects by which transformation procedures (this list is enforced by the access control system and must be certified too).

However, the Clark and Wilson model concerns a different problem domain. The approach taken in this paper may enhance a Clark&Wilson-policy by the also transaction-oriented concept of duties. On the other hand, it is not restricted to "transaction-only" systems, because it may enhance an "ordinary" discretionary access control policy too. It is not inconsistent to have an authorization scheme for read and write accesses (in the simplest case) to protection objects, besides the transaction-based concept of duties. The precondition is that also an authorization scheme for the execution of transactions exists.

Recently, some other authors have proposed an application of *normative rights* (/MoMc 91/) or *obligations* (/WiMW 89/ and /WWMD 91/), but neither of them has proposed to implement them by active mechanisms, but, of course, there may exist other ways to cope with them.

Morris and McDermid give a very elaborated distinction of the semantics of access relationships at the normative level in /MoMc 91/. They divide the process of determining the security requirements into 3 stages: an *ontologic stage* (determining the actors, possible actions, events, states of affairs or conditions, etc.), a *normative stage* (considering the type of security relationships between agents and events) and a *stage of specifications for normative conditions* (concerning the ways in which something is allowed or obliged). Their paper concentrates on the normative stage, and the duties and liberties of this paper are similar to a part of their considerations. Morris and McDermid consider 4 families (X) of rights: the *freedom* to do something, the *power* to do s.th., the *claim* to do s.th. and the *immunity* to do s.th. Each family X is further divided into 4 subclasses (simplified): X, *counter X*, *no X* and *no counter X*. It is impossible to give a comprehensive description here, but it is worth mentioning that the normative rights introduced in this paper are slightly similar to some of Morris and McDermid's rights. The main difference is that Morris and McDermid consider relationships between two actors (e.g. rights of a doctor concerning his patients and vice versa) rather than relationships between actors and data. So, both considerations concern not the same level, but it seems to be promising to investigate if a combination of both approaches is possible (e.g. a mapping between the concepts). However, it is obvious that 6 concepts (cf. Figure 1) cannot have the same expressiveness as 16. (Effectively, there are only 8 different concepts, because claims and freedoms, on the one hand, as well as immunities and powers, on the other hand, are equivalent to each other if the actors are swapped.) The following (simplified) correspondences should be taken into consideration:

<i>permissions</i>	↔	<i>power</i> and <i>not counter immunity</i>
<i>prohibitions</i>	↔	<i>not power</i> (implicit prohibition) and <i>counter immunity</i> ;
<i>positive duties</i>	↔	<i>not counter power</i> , <i>not counter freedom</i> , <i>claim</i> and <i>immunity</i>
<i>negative duties</i>	↔	<i>not freedom</i> and <i>counter claim</i>
<i>positive liberties</i>	↔	<i>freedom</i> , <i>counter freedom</i> , <i>not claim</i> and <i>not counter claim</i>
<i>negative liberties</i>	↔	<i>counter power</i> and <i>not immunity</i>

Wieringa, Meyer, Weigand and Dignum follow in /WiMW 89/ and /WWMD 91/ another approach based on deontic logic. They have introduced another kind of integrity constraints, *deontic constraints*, which could be violated by the part of the real world to be modelled (e.g. normative conditions). This leads to deontic concepts like obligations, permissions, discretions and prohibitions, which are all reduced to prohibitions and later on to actions resulting in the fulfillment of a violation condition. Their obligations are similar to (positive) duties of this pa-

per, especially with respect to the possibility that users do not obey them. Interesting is the expressiveness of their logical calculus (a combination of deontic and dynamic logic), which was proven to be sound and complete in /Wier 91/.

However, both approaches only deal with a specification of duties and not with a possible implementation.

Acknowledgements

The author would like to thank Prof. Joachim Biskup and Dr. Jimmy Brüggemann from Hildesheim University for many beneficial discussions and criticisms.

References

- /Brüg 91/ Brüggemann, H.H.; *Rights in an Object - Oriented Environment*, Proc. of the 5th Working Conference on Database Security; IFIP WG 11.3; Shepherdstown, West Virginia, USA, 3-7 Nov. 1991
- /Chak 89/ Chakravarthy, S. et al.; *HiPAC: A Research Project in Active, Time-Constraint Database Management*, Technical Report XAIT-89-02; July 1989
- /CIWi 87/ Clark, D.D. ; Wilson, D.R.; *A Comparison of Commercial and Military Computer Security Policies*; Proc. of the 1987 IEEE Symp. on Security and Privacy, Oakland, Apr. 87
- /DaBC 88/ Dayal, U.; Buchmann, A.P.; Mc Carthy, D.R.; *Rules Are Objects Too: A Knowledge Model For An Active, Object-Oriented Database System*; in: Dittrich, K.R. (ed.); *Advances in Object - Oriented Database Systems*; 2nd International Workshop on Object - Oriented Database Systems, Bad Münster am Stein - Ebernburg, FRG, Sep. 88, Proceedings
- /Daya 88/ Dayal, U.; *Active Database Management Systems*; Proc. of the 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem, Jun. 1988
- /Eswa 76/ Eswaran, K.P.; *Specifications, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System*; RH 1820 (26414), Computer Science, 1976
- /GaGD 91/ Gatziau, S. ; Geppert, A. ; Dittrich, K.R.; *Integrating Active Concepts into an Object-Oriented Database System*; Proc. of the 3rd Int. Workshop on Database Programming Languages, Nafplion, Greece, Aug. 1991
- /JoGe 91/ Jonscher D.; Gerhardt W.; *A role-based Modelling of Access Control with the Help of Frames*; Proceedings IFIP TC 11 Conf. on Information Security, Brighton (UK), May 1991
- /KoDM 88/ Kotz, A.M.; Dittrich, K.; Mülle, J.; *Supporting Semantic Rules by a Generalized Event/ Trigger Mechanism*; Proc. EDBT 1988; Lecture Notes in Computer Science 303, Springer 1988
- /Kotz 89/ Kotz, A.M.; *Triggermechanismen in DBSn*; Informatik-Fachberichte (201), Springer-Verlag, Berlin, 1989

- /MoMc 91/ Morris, Ph. ; McDermid, J.; *The Structure of Permissions: A Normative Framework for Access Rights*; Proc. of the 5th Working Conference on Database Security, IFIP WG 11.3, Shepherdstown, 1991
- /Moss 81/ Moss, J.E.B.; *Nested Transactions: An Approach to Reliable Distributed Computing*; PhD Dissertation, MIT, Cambridge, MA, 1981
- /Ston 90/ Stonebraker, M. et al.; *Third - Generation Database System Manifesto*; SIGMOD RECORD, Vol. 19, No. 3, Sep. 1990
- /Wier 91/ Wieringa, R.; *Steps towards a method for the formal modeling of dynamic objects*; Data & Knowledge Engineering, Vol. 6, No. 6. Oct. 1991, Elsevier Science Publishers B.V. (North Holland), 509-540
- /WiMW 89/ Wieringa, R. ; Meyer, J.-J. ; Weigand, H.; *Specifying dynamic and deontic integrity constraints*; Data & Knowledge Engineering 4 (1989), Elsevier Science Publishers B.V. (North Holland), 159-189
- /WWMD 91/ Wieringa, R.J. ; Weigand, H. ; Meyer, J.-J. Ch. ; Dignum, F.P.M.; *The Inheritance of Dynamic and Deontic Integrity Constraints - or: Does the boss have more rights ?*; Annals of Mathematics and Artificial Intelligence 3 (1991), 393-428

Formalising and Validating Complex Security Requirements

Philip Morris & John McDermid

University of York, Heslington, York, YO1 5DD.

ABSTRACT

In this paper we look at some complex security requirements and the difficulties which pertain to deriving a (formal) specification for them. We consider one possible approach derived from the semantic theory of the logician Rudolf Carnap. It is shown how this offers an easy solution to the problem of deriving a specification. We then consider problems of validation and whether this approach makes validation a more tractable activity. Finally we conclude by considering what further advantages this approach has.

1. Introduction

We are concerned in this paper with the process of formalising and validating security requirements. We assume the existence of an informal requirements statement written in natural language, and we investigate the issues of producing and validating a formal specification based on these requirements.

In the military security domain, many requirements are expressed in terms of information flow. We are concerned here with more general security requirements which are often couched in terms of access to resources held on computers, or access to knowledge. We are particularly concerned with complex requirements including conditional constraints on access to information or knowledge as these are difficult to formalise correctly.

Security specifications are predicates relating individuals (agents) with knowledge, or knowledge states. Thus for example, 'X knows the contents of file_one.' is an example of a knowledge state. It would be possible to introduce a logic of knowledge to represent the inferences that an individual may make having gained some particular item of knowledge. For the purpose of this paper we do not include such capabilities, but assume that knowledge of some item of information, and permission to know some item of information can satisfactorily be treated as atomic predicates. In future, it might be valuable to expand our approach by taking into account the properties of knowledge, perhaps by the use of some epistemic logic.

Security requirements can be expressed in a number of different ways. In general it is

possible to express security requirements as constraints on allowable states, and as constraints on allowable transitions. We exploit this duality in validating specifications. Our approach formalises the security specifications in terms of predicates defining allowable transitions between knowledge states, then formalising mandated and forbidden knowledge states as validation conditions. If the transitions which we have specified are consistent with the validation conditions, then we have a valid specification. This approach does not completely eliminate the 'formality gap' between requirements and specifications, but it eliminates some classes of problems.

We begin by looking at what we shall call *atomic requirements*. It will be said that a requirement is an atomic requirement just in case it specifies one normative relationship for just one agent. Otherwise the requirement will be a complex requirement. This has the effect of making "Nurses are permitted to know the contents of patient's medical files." an atomic requirement and "Nurses and Doctors are permitted to know the contents of patients medical records." a complex requirement.

Our work is motivated by this recognition that some security requirements are complex. If we try to formalise complex requirements in a 'natural' way then we can get into considerable difficulty as we find that we end up with invalid specifications. The sort of example which is motivating our work is the Chinese Wall Security Policy (hereafter CWSP). [Brewer89] We define what we consider the pertinent parts of the CWSP as follows.

Initially a user of a system is permitted to read file_one, file_two and file_three. If, however he reads file_one, then he is not permitted to go on to read file_two. If he reads file_two first, he is not then permitted to read file_one. He is permitted to read file_three regardless of whether he has read file_one or file_two.

As we shall show later a 'natural' formalisation of this problem leads to something very different from the natural language requirements. We propose a technique and a method for deriving requirements which over come this sort of problem. ⁽¹⁾

Work has been undertaken by ourselves and others [Cuppens91, Brewer89, Morris92a, Morris92b, Glasgow90] on the use of non standard logic for representing security requirements. Whilst these are appealing they also have difficulties as they are beyond the experience of many trained computer scientists. The approach we propose here shows a way of using standard first order logic to formalise security requirements without falling into logical problems. We return to this below.

(1) Although the example we have chosen is small, there is no reason why the method advocated should not be applied to larger examples, such as Ting's Medical example. [Ting90]

2. Complex Requirements

While defining the specification for a requirement, it is important that we define the relationships which exist between the various requirements so that the conditions under which the requirements obtain are made clear. It is easy to show why.

Suppose we have two requirements "Doctors are permitted to know the contents of patient's medical files." and "If doctors are permitted to know the contents of patient's medical files then nurses may be given permission by doctors to read patient's medical files.". If we ignore the distinction between complex and atomic requirements and do not heed the logical structure, i.e., the relations between atomic requirements, then our specification will not sanction the sorts of inferences we require, as we shall now illustrate.

Let 'p' stand for "Doctors are permitted to know the contents of patient's medical files.", 'q' stand for "If doctors are permitted to know the contents of patient's medical files then nurses may be given permission by doctors to read patient's medical files." and 'r' for "Nurses may be given permission by doctors to read patient's medical files." The formalisation for the requirement would be 'p & q'. And from this we would not, by the rules of inference for propositional logic, be able to infer 'r', i.e., we would not be able to infer, from the formalisation that nurses may be given permission to read patient's medical files. This would then be a severe limitation on our specification language, not least because it makes validation that much more difficult given that we do not know what some of the entailments from the specification are. It is therefore important that the formalisation uncovers the logical structure of the natural language requirements.

There is an additional problem with writing specifications for complex requirements. This concerns not so much a failure to distinguish atomic from complex requirements, but rather a failure to capture the types of logical relationships which exist between the requirements. Consider again the CWSP. This states that a user is initially permitted to read file_one, file_two and file_three. If he reads file_one, then he is not then permitted to read file_two, if he reads file_two, then he is not permitted to read file_one. Let 'p' stand for 'X is permitted to read file-one', 'q' for 'X is permitted to read file-two' and 'r' for 'X is permitted to read file-three'. (We shall use the letters to stand for these statements throughout the paper.) We can now specify the Chinese Wall Security Policy in the obvious way, by formalising each natural language statement as 'p & q & r & (p \rightarrow \neg q) & (q \rightarrow \neg p)' then from this we can infer (\neg p & \neg q). (For the proof of this see Appendix I.) The intuitive but incorrect formalisation leads us to the absurd position of denying access to both file_one and file_two. This is clearly undesirable. The source of the error, in this example, is obvious, but it may not always be so.

This suggests that a procedure is required for unravelling the complex requirements in such a way that we ensure the correct formal specification.

Validation is also an important aspect of requirement specifications and has implications

for what the requirements specification should look like. (A specification that cannot be validated will be of no use.)

In this paper we concentrate on validation by considering the possible states which the system might be in, rather than considering the transitions which the state might go through. Although this is not argued for in this paper, it is not clear why a security requirement cannot be validated solely by considering states rather than by states and possible state transition, if only because the authors believe the method that is being proposed allows us to infer which state transitions are permitted and which not. ⁽¹⁾

3. State Descriptions

What a specification for a complex requirement should look like is not always obvious. The 'logic' which lies beneath natural language constructions is not always perspicuous. Some procedure is required which will ensure the specification is the most appropriate one for the requirements.

In his book 'Meaning and Necessity' Carnap [Carnap47] introduces the notion of a *state description* for a formal language (such as propositional logic) S_I :

"A class of sentences in S_I , which contains, for every atomic sentence either this sentence or its negation, but not both, and no other sentence, is called a state description in S_I ..."

And a state description is useful because

"... it obviously gives a complete description of a possible state of the universe of individuals with respect to all properties and relations expressed by the predicates of the system. Thus the state descriptions represent Leibniz' possible worlds or Wittgenstein's possible states of affairs."

Now suppose we substitute "atomic requirement" for "atomic sentence". We can view our requirement as a class of atomic requirements. Thus the CWSP will have three basic requirements: {X is permitted to read file_one, X is permitted to read file_two, X is permitted to read file_three}.

We can form a state description of a system by conjoining *every* atomic requirement, or its negation, but not both. This will define one possible state description. Thus, in the CWSP, "X knows the content of file_one and X does not know the content of file_two and X knows the content of file_three." is one state description.

We define all the possible state descriptions from the set of atomic requirements by taking the power set of that set. This will give us a set of partial and complete state descriptions.

(1) If, for example, $(p \ \& \ \neg q \ \& \ \neg r)$ and $(p \ \& \ \neg q \ \& \ r)$ are both permitted states, then the transition from $(p \ \& \ \neg q \ \& \ \neg r)$ to $(p \ \& \ \neg q \ \& \ r)$ would be a permitted transition.

We form all the possible state descriptions from the partial ones by supplying the negated requirement where it is absent from the set.

Thus the power set of the set $\{A,B\}$ would be $\{\{\},\{A\},\{B\},\{A,B\}\}$. We define all the state descriptions from the three partial ones by supplying the negation of the missing requirements. This will give us: $\{\{\neg A, \neg B\}, \{A, \neg B\}, \{\neg B, A\}, \{A, B\}\}$ which would then give all the possible combinations.

This is all the *logically possible* states. These are not going to correspond to the customer's requirements. The customer is only going to want some of these states to obtain. We need to determine which are consistent with the requirement.

This is done by constructing a table. Let us call this a state description table. Each row on the table represents a possible state description. Each state description can be checked against the customer's requirement.

We can work through the table row by row to build up a complete picture of all the possible states. From the complete state description we can work out the requirement specification by considering which of those states is consistent with the customer's requirements.

Consider again the CWSP. The complete state description for these atomic requirements are given as in the above table. (Where 'p', 'q' and 'r' are defined as above.)

	p	q	r	State Descriptions
(1)	T	T	T	F
(2)	T	T	F	F
(3)	T	F	T	T
(4)	T	F	F	T
(5)	F	T	T	T
(6)	F	T	F	T
(7)	F	F	T	T
(8)	F	F	F	T

To the right of the table, underneath the heading 'state description' we assign a value T or F in accordance with whether or not that state description is consistent with the requirements. Thus in the first row the state description takes the value F since the description 'X knows the content of file_one and X knows the content of file_two and X knows the content of file_three' is not consistent with the requirements.

Note that (4), (6) and (8) can be true because it is consistent with the requirement that X does not know the content of file_one, two and three. (Although it is interesting to note that if we define the requirements in terms of preconditions, or actions, then (4), (6) and (8) will be false.)

We can derive from this a complete description of all the possible states which the system could be in and which are therefore consistent with the requirements by conjoining the six true formulae.

$$(p \& \neg q \& r) \vee (p \& \neg q \& \neg r) \vee (\neg p \& q \& r) \vee (\neg p \& q \& \neg r) \vee (\neg p \& \neg q \& r) \vee (\neg p \& \neg q \& \neg r)$$

where for example, the first disjunct represents row (3), the second row (4) and so on. We can use a Karnaugh Map to reduce these to:

$$((p \& \neg q) \vee \neg p).$$

which gives us the required specification ⁽¹⁾. This therefore gives us a way of capturing and specifying complex requirements. We now need to see how this relates to the validation conditions.

4. Validation Conditions

We begin by considering what is meant by, or what is involved in, the concept of validation. We mentioned above the possibility of looking at each state that is consistent with the specification. It is obvious that validation cannot be done by checking the set of all possible states, or state transitions, which are consistent with this, since that could be an infinite number.

Rather we consider two subsets of combination of requirements; those combinations which ought to always occur, and those which ought never to occur. Call the former **mandated states** and the latter **forbidden states**. For more discussion on these see [Dobson90]

Mandated and forbidden states are something we can enumerate and reason about, since they are finite. They will, for example, take such forms as: State A ought always to occur with state B, and state B should never occur with state C.

Rather than trying to validate each state or state transition that is consistent with the specification against the requirements, it would be better to *prove that every state that is consistent with the requirement always has the mandated properties, and never has the forbidden properties.*

(1) It may come as a surprise that r does not appear in the final specification. However, since r ('X knows the content of file_three') and $\neg r$ ('X does not know the content of file_three.') are both consistent with any permitted permutation of 'p' and 'q', it follows that were r to be included the specification would in fact be $((p \& \neg q) \vee \neg p) \& (r \vee \neg r)$ which is, of course, logically equivalent to the above specification. Although we have not argued for this in this paper we believe that the consequence of this is that the requirement ' r ' is independent from 'p' and 'q'.

We need first to be clear about what it is for a state to be consistent with a specification, what it is for a state to be mandated, and what it is for a state to be forbidden.

4.1. Consistency

A state c is said to be consistent with a specification S just in case it is not possible for c to occur without S occurring. This is a strong conception of consistency. But it has an important advantage over other weaker conceptions. ⁽¹⁾ It means, for example, that although 'X has knowledge of file_one and three and not file_two.' is consistent with the specification, 'X has knowledge of file_three.' is not. ⁽²⁾

The motivation behind this is that, for example, "X has knowledge of file_three." cannot be said to be consistent with the specification because we are unclear what the truth of 'X knows the contents of file_two.' and 'X knows the contents of three.' are. This strong conception has the advantage of refusing to acknowledge that an incomplete (state) description can be consistent with a specification. And this is correct, it should refuse to do this precisely because we do not know the values of the omitted atomic requirements.

More formally any state c is said to be consistent with S just in case

$$\vdash c \rightarrow S$$

4.2. Mandated States

Although we refused to acknowledge that partial state (descriptions) can be consistent with a specification, mandated states can be partial. Thus, for example 'If X has knowledge of file_one then X does not have a knowledge of file_two.' can define a mandated state even though there is no mention of X's relation to file_three.

State m will be said to be a mandated state of a specification S just in case every time S is true, m is also true. We express this more formally by saying:

$$\vdash S \rightarrow m$$

Given the above specification it can be shown that 'If X has knowledge of file_one then X does not have a knowledge of file_two.' is, by our definition, a mandated property since the following holds:

$$\vdash (((p \ \& \ \neg q) \vee \neg p)) \rightarrow (p \rightarrow \neg q)$$

(The proof of this is to be found in Appendix II.)

(1) A weaker definition would be c is consistent with S just in case $\neg \vdash S \ \& \ \neg c$

(2) Whereas it would be consistent under the above weaker definition of consistency

4.3. Forbidden States

Just as some requirements have persistent properties so some will also have forbidden properties. Forbidden properties, like mandated states, can be partial. An example of a forbidden state (again from the CWSP) is "X has knowledge of file_one and X has knowledge of file_three."

A state f is said to be a prohibited state for a specification S just in case f never occurs. We express this formally by saying:

$$\vdash S \rightarrow \neg f$$

It is not difficult to see why this works. This states that whenever the specification is true, the state or partial state f will be false.

4.4. Mandated States and Consistency

If part of the validation consists in showing that the mandated states obtain for a specification, then we will need to prove that the mandated states will obtain for every state that is consistent with the specification, i.e., we need to prove the following:

$$\vdash (((c \rightarrow S) \& (S \rightarrow m)) \rightarrow (c \rightarrow m))$$

That is to say, for any state c which is consistent with the specification S and any state m which is a mandated state for the specification, will also be a mandated state for any state that is consistent with the specification. (The proof of the theorem is given in Appendix III.)

4.5. Forbidden States and Consistency

Similarly we need to prove that any state c which is consistent with specification S will not have any forbidden properties f . That is to say:

$$\vdash (((c \rightarrow S) \& (S \rightarrow \neg f)) \rightarrow (c \rightarrow \neg f))$$

The proof to this is given in appendix IV.

This proof can be exemplified by considering another example from the CWSP. Let 'p', 'q' and 'r' stand for the security requirements as above. It is clear that $(p \& \neg q \& r)$ is a state description which is consistent with the specification since $\vdash ((p \& \neg q \& r) \rightarrow ((p \& \neg q) \vee \neg p))$. Furthermore $(p \& q)$ is a forbidden state since $\vdash ((p \& \neg q) \vee \neg p) \rightarrow \neg(p \& q)$. $(p \& q)$ is also forbidden for the consistent state $(p \& \neg q \& r)$ since $\vdash (p \& \neg q \& r) \rightarrow \neg(p \& q)$.

5. Validation

An ideal aim of validation would be to show that each state s that is consistent with the specification is what the customer requires. We argued this was not always possible. But what could be done was to show that the specification only sanctioned states which preserved all mandated states and precluded forbidden states.

It was claimed that to do this we would need to prove that all states which were consistent with the specification preserved the mandated states and precluded the forbidden states. This we have done. Thus it follows that any state mandated or forbidden by the specification will also be mandated or forbidden by any state consistent with the specification.

For validation of requirements, therefore, all that is required is to ensure that the mandated and forbidden states entailed by the specification are in accord with the customer's requirements.

6. Conclusion

In this paper we have given an all too terse account of a method for formalising and validating security requirements. As a conclusion we should like to make a few brief additional points concerning the method we have been advocating.

We have talked about complex (security) requirements, how we might derive a formal specification for them, and what would constitute a validation. Validation, it was said, consists in proving that all mandated and no forbidden states obtain. Yet this might not be enough to gain assurance. Are there any other properties of the state description tables we might be used for considering validation.

One useful concept might be that of independence. It is not difficult to envisage a situation where what is required is a proof that the truth (or falsity) of this state has no logical or causal bearing upon the truth (or falsity) of that state. (There is an example of an independent requirement in the CWSP. The truth (or falsity) of "X has knowledge of the contents of file_three." has no logical or causal bearing on the truth (or falsity) of either "X has knowledge of the contents of file_one." or "X has knowledge of the contents of file_two.")

The authors believe that the proposed method will give some grounding to this conception, and allow us to prove, for any specification, which requirements are independent from each other. Although we do not consider it in detail here it suffices to say that we consider two requirements to be independent from each other just in case there are state descriptions in which all four possible truth combinations can be found.

One difficulty with this approach might be that the state description tables are too large. Although this does not constitute a logical objection, it does make the process of formalising the requirement that much more difficult. The concept of independence will also have

some benefit here. It will considerably ease the process of securing a specification by reducing the size of the state description tables, in fact we will only need to construct tables for those requirements which stand in a conditional relationship to each other. The authors also believe, however, that they have uncovered a method which solves the problem of scalability and does not require working with large state description tables; indeed the tables need never have more than four lines.

Just as there can be independence between requirements, so there can be logical relations. Again these prove useful in attempting to gain assurance that the specification does what is required. An important part of validation might, for example, consist in showing the conditions under which a state, or a partial state obtain. We might want to know whether, according to the specifications, having knowledge of `file_one` is a sufficient condition for not having knowledge of `file_two`, or whether there are any conditions under which a user cannot have knowledge of `file_three`. Again we do not intend, in this paper, to go into details here, but suffice it to say that the method advocated permits an automatic derivation, for every atomic requirement, of all the necessary and sufficient conditions which are required for that state to obtain.

In addition to plotting logical relations between various requirements, it would also be useful to derive possible causal relations between the requirements using the state description tables. We are, at the moment working on this and considering what, if any, additional information, is required which would enable us to do that.

Finally, in the introduction to this paper it was stated that it might be valuable to extend the approach by taking into account the properties of knowledge, perhaps by the use of an epistemic logic. It is not too difficult to see how this might be achieved. Propositional variables in the state description tables, would be replaced by formulae, either from an epistemic logic [Cuppens91, Glasgow90] or a deontic specification which captures the types of access rights which are involved [Morris92a, Morris92b]. Which gives us a greatly enriched specification.

There is still much to be done with the state description approach, but we believe that it offers an easy and promising way of deriving a formal specification from complex security requirements and which also enable us to validate the specification against the requirements.

- Brewer89. Brewer, David F.C., & Nash, Michael, J., "The Chinese Wall Security Policy", in *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press (1989).
- Carnap47. Carnap, R., *Meaning and Necessity. A study in Semantics and Modal Logic*, University of Chicago Press (1947).
- Cuppens91. Cuppens, F., "A Modal Logic Framework to Solve Aggregation Problems", in *Proceedings 5th IFIP WG 11.3 Conference on Database Security* (1991).
- Dobson90. J. E. Dobson and J. A. McDermid, "An Investigation into Modelling and Categorisation of Non-Functional Requirements (for the Specification of Surface Naval Command Systems)", YCS 141, Dept. of Computer Science, University of York (Aug 90).
- Glasgow90. Glasgow, J., McEwen, G., & Panangaden, P., "A Logic for Reasoning About Security", *Proceedings of the Computer Security Foundations Workshop*, Franconia (1990).
- Morris92a. Morris, P., & McDermid, J., "The Structure of Permissions: A Normative Framework for Access Rights", in *Proceedings of the Fifth IFIP WG11.3 Conference on Database Security*, ed. Landwehr, C.E., Springer Verlag (1992).
- Morris92b. Morris, P., & McDermid, J., "The Structure of Permissions: A Normative Framework for Access Rights", in *Proceedings of the Fifth IFIP WG11.3 Conference on Database Security*, ed. Landwehr, C.E., Springer Verlag (1992).
- Ting90. Ting, T.C., "Application Information Security Semantics: A Case of Mental Health Delivery", in *Database Security: Status and Prospects III*, ed. Spooner, D.L., & Landwehr, C.E., North Holland, Amsterdam (1990).

Appendix I

- | | | |
|-----|---|---------|
| (1) | $p \& q \& r \& (p \rightarrow \neg q) \& (q \rightarrow \neg p)$ | A |
| (2) | p | 1 &E |
| (3) | $q \rightarrow \neg p$ | 1 &E |
| (4) | $\neg q$ | 2,3 MTT |
| (5) | q | &E |
| (6) | $p \rightarrow \neg q$ | &E |
| (7) | $\neg p$ | 5,6 MTT |
| (8) | $\neg p \& \neg q$ | 4,7 &I |

(It should be noted that there are other horrors with this specification. From lines (4) and (5) we can of course infer $(q \& \neg q)$. Since this is a contradiction, anything follows from it.)

Appendix II

$\vdash (((p \& \neg q) \vee \neg p) \rightarrow (p \rightarrow \neg q))$

- | | | |
|-----|---|--------------|
| (1) | $\vdash (p \& \neg q) \vee \neg p$ | A |
| (2) | $p \& \neg q$ | A |
| (3) | p | 2, &E |
| (4) | $\neg q$ | 2, &E |
| (5) | $p \rightarrow \neg q$ | 3,4 CP |
| (6) | $\neg p$ | A |
| (7) | $\neg p \rightarrow ((p \rightarrow \neg q) \rightarrow \neg p \rightarrow (p \rightarrow \neg q))$ | |
| (8) | $p \rightarrow \neg q$ | 6,7 MPP |
| (9) | $p \rightarrow \neg q$ | 5,8 \vee E |

(To keep the proof simpler we have, at line (7) used a theorem from the propositional calculus.)

Appendix III

Mandated Properties and Consistency

$\vdash ((c \rightarrow S) \ \& \ (S \rightarrow m)) \rightarrow (c \rightarrow m)$

- | | | |
|-----|-------------------|---------|
| (1) | $c \rightarrow S$ | A |
| (2) | $S \rightarrow m$ | A |
| (3) | c | A |
| (4) | S | 1,3 MPP |
| (5) | m | 2,4 MPP |
| (6) | $c \rightarrow m$ | 3,5 CP |

Appendix IV

Forbidden Properties and Consistency

$\vdash ((c \rightarrow S) \ \& \ (S \rightarrow \neg f)) \rightarrow (c \rightarrow \neg f)$

- | | | |
|-----|------------------------|---------|
| (1) | $c \rightarrow S$ | A |
| (2) | $S \rightarrow \neg f$ | A |
| (3) | c | A |
| (4) | S | 1,3 MPP |
| (5) | $\neg f$ | 2,4 MPP |
| (6) | $c \rightarrow \neg f$ | 4,5 CP. |

Support for Security Modeling in Information Systems Design

Gerhard Steinke

University of Passau, Innstr. 33, 8390 Passau, Germany

Matthias Jarke

Informatik V, RWTH Aachen, Ahornstr. 55, 5100 Aachen, Germany

Abstract

We present a set of modeling constructs and reasoning tools that extend the use of computer-supported conceptual modeling for information systems to the study of security aspects. The modeling framework is the Group Security Model (GSM) which describes access rights through a teamwork-oriented organizational model. Reasoning about GSM application models is enabled by representing them in a deductive and object-oriented database language, Telos. A prototype implementation in the software information system ConceptBase is reported.

1. INTRODUCTION

Conceptual modeling has been widely applied in information systems requirements engineering and design. While this was mostly a pencil-and-paper exercise for quite a while, a number of tools have recently become available through which conceptual models can be formally represented as knowledge bases and analyzed with knowledge base tools [Borgida et al. 87].

Initial conceptual models just specified the internal structure of systems. More recently, the emphasis has shifted to describing the relationship of the information system to its environment. According to a model proposed in [Jarke 90], this environment can be categorized into the usage, the subject, and the development world of the system. The usage world describes the organization in which the system is intended to function, the subject world describes the part of the world the system maintains information about, and the development world describes the version history of system components and development teams. Each of these subworlds, as well as their interactions, create certain nonfunctional requirements which have to be taken into account when making or evaluating design decisions about the system. Domain theories can be developed which provide conceptual frameworks and reasoning tools for representing and using such requirements.

This paper presents such a domain model for the goal of information systems security in an organizational setting. Security can be understood as a type of nonfunctional requirement which constrains the relationship "access rights" established by the information system between the usage world and the subject world. Our conceptual model, the Group Security Model (GSM), assumes that an information system is embedded in a teamwork-oriented usage environment where teams have to accomplish certain tasks and *therefore* need certain information.

The GSM is a framework for discretionary security policies with some mandatory aspects. It is mandatory in the sense that certain organizational and access right structures are defined in a centralized fashion at information systems design time, and that users may transfer rights only within this framework. However, as in discretionary models, there is no central definition of classification levels for users but a concept of subsystem ownership and distributed control. As a further note to security specialists, we should say that we take a fairly naive view of access rights here: we assume a safe underlying encryption and communication system and ignore the dangers of hidden channels (but see [Steinke 91] for a discussion of one such channel, the handling of integrity constraints).

There are several possible uses of such a model. First, if designers choose the GSM framework for information systems engineering and design, they can use the associated analysis toolkit to study security aspects of the proposed system interactively. Through query mechanisms, they can simulate the system behavior and find out who has access to what objects. Through limited theorem-proving capabilities from the domain of semantic query evaluation, they can study which task (and thus access right) combinations should be avoided from a security standpoint. Through tracing mechanisms, they can obtain explanations how users have obtained or could obtain implicit access to objects.

Similarly, the impact of proposed organizational and design changes in existing systems can be analyzed, and security-oriented re-engineering of existing information systems can be supported when a reverse engineering study of them is made within the GSM framework.

A toolkit for reasoning about GSM-based security is contributed by representing the GSM in a knowledge representation language, Telos [Mylopoulos et al. 90], that combines features of deductive and object-oriented database languages. The security modeling primitives (such as classes of access rights) are defined in this model as metalevel deduction rules and integrity constraints which are then compiled using optimization techniques from deductive databases offered by the software information system, ConceptBase [Jeusfeld and Jarke 91, Jeusfeld and Staudt 91].

The GSM was originally proposed for the context of security in arbitrary multi-user knowledge bases. However, due to the computational problems associated with theorem-proving in complex rule systems we expect our tools to be most practically relevant in the formally rather simple object-oriented knowledge bases used to describe information systems requirements and designs. We therefore focus on this aspect here.

Section 2 gives an overview of previous work on reasoning about security. Section 3 describes the GSM framework and section 4 the associated analysis toolkit. Section 5 provides comments on the implementation as well as suggestions for further research.

2 BACKGROUND AND MOTIVATION

Reasoning is the ability to draw conclusions and create information by applying rules and functions on prior knowledge, in addition to the capability to provide an explanation and justification of specific actions and conclusions.

Reasoning about knowledge, based in the philosophical tradition, has become an active area of research in a wide variety of fields ranging from artificial intelligence and cryptography to linguistics and psychology. Reasoning about knowledge is also a basic feature and capability of knowledge base systems. Knowledge base systems create information based on the triggering and evaluation of rules by the control structure. A "characteristic of knowledge-based systems is that they are (ideally) able to reason about their own reasoning process" [Berson and Lunt 87, p.237]. The capability of this reasoning process should be made available to users in order to help them understand and improve the design, implementation and operation of the security component of a knowledge base system.

U-Kuang is a rule-based tool which performs reasoning about the security of UNIX systems [Baldwin 87]. U-Kuang deduces the set of operations directly or indirectly accessible to each user, particularly those operations which can extend a user's privileges. In order to determine these operations, U-Kuang contains information about UNIX protection modes, files, privileged programs and files they access, etc. U-Kuang determines the potential operations accessible to users and compares the operations with the specified security policy which contains all users and their allowed user and group privilege. The operations which violate the security policy are reported.

The application of reasoning about security is often considered to be only an optional or nice, rather than essential security component. Although reasoning about security does not provide security in itself, it enables the security of a system to be examined and evaluated. However, for information engineering in organizations, the possibilities for security strongly depend on the organizational design that implies the "need-to-know".

[Glasgow et al. 91] pursue a somewhat similar point of view when they define reasoning about security in terms of the interacting notions of knowledge, obligation and permission. They create a structured framework for reasoning about security based on a modal logic formalism, as well as a language for expressing abstract security policy definitions. However, no practical reasoning tools are readily available for such special-purpose logics.

One major issue in reasoning about security has been complexity. In the discretionary policies we are interested in here, the early results were either negative (undecidability of the access matrix model [Harrison et al. 1976]) or led to overly simplistic models. However, concepts from the database area, in particular the ideas of database schema and of transaction specification, have recently led to more realistic solutions.

The one closest to our approach is the Schematic Protection Model (SPM) by Sandhu [1988]. Security policies can be defined by a meta database schema which defines permitted relationships between models of active components (*subjects*) with each other and with passive *objects*. The complexity of analysing whether -- under a given initial database state -- a given subject could potentially access a certain object then depends mostly on the way how transfer rights (copy flags) can be transferred among subjects. Specifically, Sandhu identifies two sufficient conditions for polynomial complexity of potential access analysis:

- (1) *Acyclicity*: if subjects of type A create subjects of type $B \neq A$ directly or indirectly, then subjects of type B must not create, directly or indirectly, subjects of type A . Cycles within the same type are allowed.
- (2) *Attenuation*: when a subject a (of type A) creates another subject b (also of type A), then b must have equal or less rights than a . This implies that a can only transfer its own rights at creation time and that any new right b receives later, is also given to a .

In related work, Sandhu demonstrates that most practically relevant protocols in the literature can be reformulated to satisfy these conditions. In particular, the coverage of hierarchically organized groups such as formal organizations is possible as well as the consideration of different user roles (such as leader and different levels of members).

In summary, Sandhu's models provide a thorough study of how the relationships between subjects and objects should be constrained such that security analysis becomes practically feasible. The basic idea is shown in figure 1.(a).

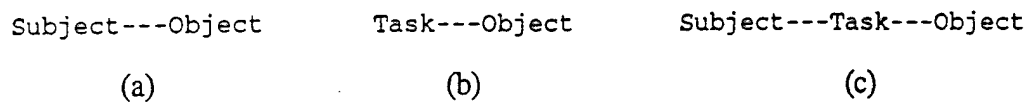


Fig. 1: Security modeling, information systems engineering, and a synthesis.

Unfortunately, this is not exactly how standard methods for organizational information systems analysis and design proceed. Most of them (including, e.g., IBM's well-known Business Systems Planning) identify the required relationships of business processes or tasks to data objects and would ignore the people or groups involved. See figure 1.(b).

Yet, it is individuals (or small groups thereof) who pose the major threats to security; and often these threats stem from the combination of different tasks (and thus justified access rights) addressed by the same organizational unit. The approach described below can be seen as attempting a synthesis between the two viewpoints as shown in fig. 1.(c). From the viewpoint of security analysis, it provides an answer to the question: *where do the subject groups mentioned in security models come from?*

From the viewpoint of information systems analysis, it allows the inclusion of security aspects without much extra data collection work for the analyst. More importantly, it also allows *organizational design for security* in the sense that proposed partitionings of task assignments to organizational units can be analyzed for their security implications. Finally, it should be mentioned that the same conceptual structure also turns out to be useful as a basis for computer-supported cooperative work [Rose et al. 92].

Obviously, to make this idea fully operational, such an organizational model must be mappable to a formal security model such as SPM. In particular, the acyclicity and attenuation properties of a system that contains multiple kinds of interwoven "active components" (subject organizations and task structures) need to be studied. In the following exposition, we do not yet solve this problem completely but offer the following two contributions: a domain analysis of what features and properties the proposed model needs to be useful both from a general information engineering and from a security perspective, and an implemented toolkit that employs deductive query processing facilities for static and partial dynamic security analysis.

3 THE GROUP SECURITY MODEL

The goal of achieving a secure system is approached by restricting a user to a subset of the information in the system. The smaller the subset of the information which a user may access, the less information that can be disclosed or modified by the user, resulting in a more secure system. The smallest subset of information which must be made available to a user consists of that information which the user needs to know. In addition, one can also restrict the operations which a user may carry out w.r.t. the information which he may access. These operations or access types would be defined according to what the user needs to do with the accessible information.

Both the need-to-know and the need-to-do are not just determined by a user's membership in a formal organizational unit but much more precisely by the tasks he or she is involved in. The Group Security Model (GSM) [Steinke 92] restricts potential user access rights to task-related information while leaving actual access right granting at the discretion of the data or procedure owner.

The GSM is formally defined as a structured semantic network in the deductive and object-oriented database language Telos [Mylopoulos et al. 90]. Telos supports the abstraction principles of classification/ instantiation, generalization/specialization, and aggregation/ attribution. Specifically, the instantiation hierarchy leads to the following layered representation of security-related knowledge (the same is, incidentally, true for other aspects of IS modeling):

- The meta meta class level defines the generic GSM framework.
- The meta level class specifies particular kinds of task-based security policies within the framework.
- The simple class level describes the security model of a particular information system within a meta level policy.
- The instance or token level is a record of actual user roles and accesses within an information system which could be used, e.g., as an audit trail.

Each level can be seen as a type system that provides a sublanguage which constrains what can be defined at the level below. The security analysis toolkit currently focuses on the lowest two levels, making certain assumptions about the policies defined at the meta level.

The two highest levels of the GSM is represented in fig. 2 as a semantic network, using graphical standards of Telos. The upper box describes the meta meta level of the model, the lower one a set of specific meta level constructs we have found useful for our prototype. In the figure, thin links with no labels stand for instantiation relationships, thick grey links for specialization relationships and the labeled links are attributes. Rectangles represent class objects and ovals represent pre-defined simple class objects which are needed in every GSM-based information systems model.

At the metalevel the GSM consists of five basic components: user, role, task, access type and object. The entity *user* represents the individuals who have permission to access the information system. A *task* represents the activities and assignments which are to be carried out in an organization. The entity *object* represents information in the system. A *role* is a link between *user* and *task* which represents the relationship that a user has to a task, e.g., leader and member. An *access type* is a link between *task* and *object* which represents the operations that a user associated with the task may carry out w.r.t. an object, e.g., read, write, and owner.

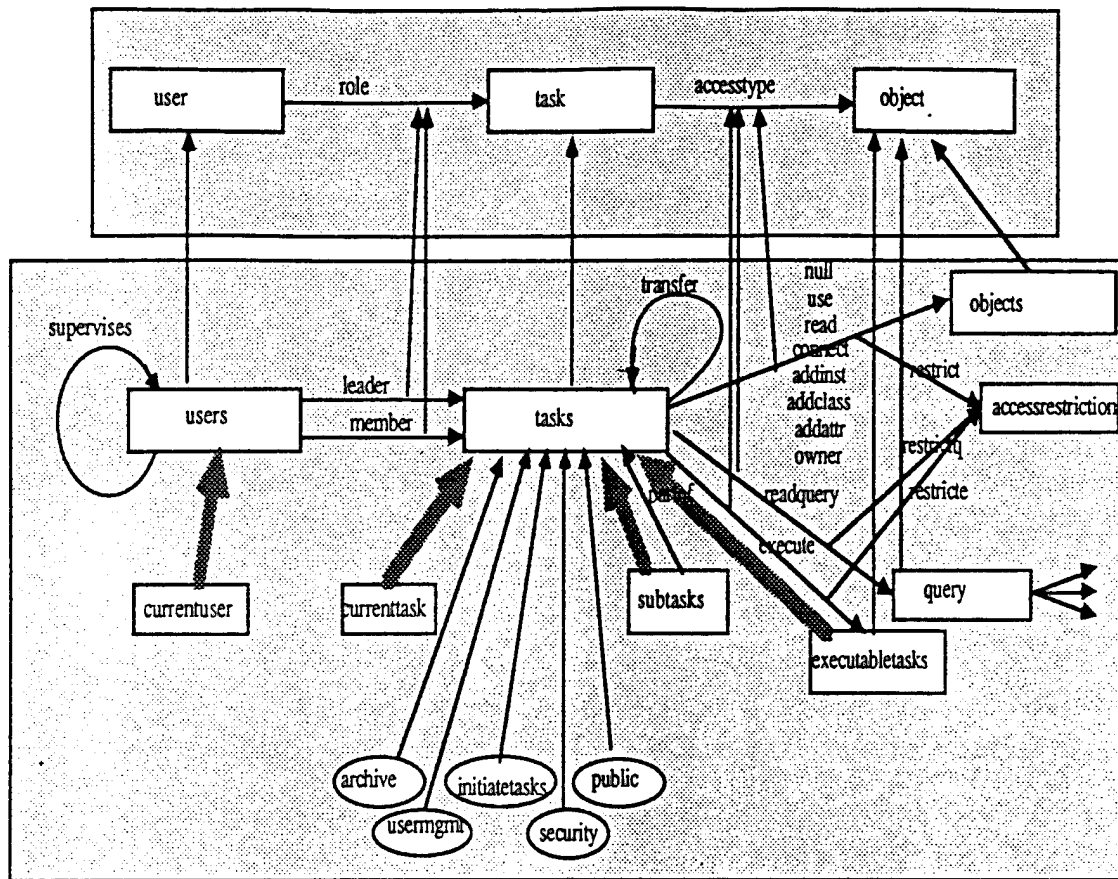


Fig. 2. Group Security Model.

A user is assigned to a task by means of a role link. Information is assigned to a task by means of an access type link. Thereby a user is restricted to that information he needs to access and to those operations he needs to do in order to complete his assigned task.

Besides the above-mentioned abstraction principles, Telos offers deductive rules and integrity constraints as in deductive databases as a basis for formally defining the semantics of concepts. Thus, the GSM and different kinds of policies within the GSM framework can be formally specified by a combination of pre-defined structure (as shown above) with rules and constraints. The following provides an informal overview of the characteristics and implications of the components of the GSM; the full formal definition can be found in [Steinke 92].

User. The object user in the GSM represents an individual or process which has permission to access the information system. Users may be assigned to more than one task, just as in the real world a user may be required to complete several assignments at the same time. The GSM specifies, though, that a user may only access information associated with a single task at a time, since all information required for a task is associated with that task. Accessing information from several tasks at the same time goes beyond the need to know principle.

The GSM enables supervisor-subordinate relationships between users to be represented by means of the supervises link. The leader of a task may only assign the users which he supervises to his task.

Role. A role defines the relationship which a user has w.r.t. a task. Two types of roles are defined in the GSM, member and leader.

The member role relationship between a user and a task permits a user to access all objects related to the task with the access type specified for that task, except for one restriction. The member role does not permit communication with other tasks, and therefore does not include the right to grant another task access permission to information owned by this task.

The leader role provides a user with responsibilities and privileges in addition to those granted to a user who is a member of the task. While every task must have a leader, it is possible for a task to have more than one leader, in which case each leader has the same rights and privileges. Only the leader of a task may communicate with other tasks. A leader is able to grant and revoke access permission to objects owned by this task. A leader has the capability to define subtasks and assign users to his task or to his subtasks.

Task. Task is the central component of the GSM. A task represents an activity or assignment which, in the process of being carried out, requires access to information in the system. There are two types or classes of tasks: first level tasks and subtasks. First level tasks are explicit instances of the class tasks representing tasks which are not subtasks of other tasks, i.e., have no parent task. Subtasks are instances of the subtasks class, created by the leader of a task, and have a partof link to the creating task, i.e., their parent task.

A transfer link, from one task to another task, indicates that access to information which is owned by the former task may be granted to the latter task.

A number of task classes are required by the GSM for administrative purposes. The public task is a task which has associated with it information which all tasks may access. The initiate tasks task allows a user associated with this task to create and terminate first level tasks. A user with a role in the user mgmt task may add and delete users and supervises links in the GSM. The archive task is the owner of all objects which have no owner, e.g., where the task which was the owner has been terminated. The current task is a subclass of tasks which contains only one instance, namely the task which is currently accessing the information system. A user associated with the security task may use the security analysis toolkit described in the next chapter.

Access Types. The access type component of the GSM, represented by a link from a task to an object, describes the operations that a user associated with the task may perform on the object. There are several ways an access type is created. An owner access type link is granted to the task which creates the object. Other access type links are generated by the owner of an object when granting access permission to his object to another task. In addition, access type links are created by leaders of tasks copying them from a parent task to a subtask. Access permissions may also be revoked by the owner of the object.

Access restriction represents a constraint which may be associated with an access type. This constraint would describe a condition which must be satisfied before the access type is used to permit access to an object. The access restriction is created by the owner of the affected object.

The number of access types is dependent on the security policy and the knowledge representation language. For Telos-based information system descriptions, the following access types have been defined [Steinke 92]:

Null access type for an object means that no access to this object is permitted. While the absence of an access type link also indicates that access to an object is not permitted, null access type reinforces the denial of access, including the denial of indirect or implied access permission, and overrides all other access types.

Use access permission for an object allows the object to be retrieved from the knowledge base for use within the system, but prevents the object from being displayed on an output device. Use access of a rule or constraint permits the rule or constraint to be used for deductions and integrity checking, while the content of the rule or constraint remain hidden from the user.

Read access for an object means that the object may be retrieved from the knowledge base. The GSM defines read access for a link object to imply read access to both the source and the destination objects. In Telos this applies to attribute links, instance of links and isa links.

Connect access for an object means that this object may be specified as the destination of a link. If read access was sufficient for an object to become the destination of a link, then —due to referential integrity of the database — the owner of the destination object may not be able to delete his object, even though the only permission he granted was read access.

Addinst access for an object provides permission to create an instance of that object. The task which creates the instance becomes the owner of the new object.

Addattr access for an object means that one has permission to add attributes to the object. If the attribute is an instance of an attribute category then one requires addinst access to the attribute class object. If the destination of the attribute is a specific object in the knowledge base (i.e., not integer or string), then connect access to that object is required as well.

Addclass access type permits the creation of a subclass, and is therefore only appropriate for a class object. If a subclass is created, additional rights are granted to the owner of the subclass w.r.t the parent classes and their attributes, in order to allow inheritance from the parent classes.

Execute access for an object provides permission to initiate a method or process.

Owner access type indicates the task which is the owner of the object. As owner, a task has the capability to perform any operation on that object. Owner access contains the privilege of granting or revoking access permission for the object, to or from other tasks. The owner may delete his objects.

To enable inheritance, owner access of a class object includes additional rights. The owner of a class object has access permission to objects "below" the class object, e.g., all instances of the class object, all attributes of the instances, as well as all subclasses of the class object (but not the attributes associated with the

subclasses). In addition, the owner of a class object may access objects "above" the class object. This allows the owner of a class to use the normally inherited attribute categories as well as recognize that instances of his class are also instances of the parent classes.

Read Query access type from a task to a query object results in access permission to a group of objects, namely all objects which satisfy the specifications of the query, at the time that the access is checked. The read query access permission can only be created by the leader of the task which has owner access to all components of the query definition, since knowledge of the objects specified in the query may be provided to the recipient of the query access. Read access permission is required for the query object itself to be accessed, i.e., to see the makeup of the query.

Objects. The object component of the GSM represents an entity or object in the information system, be it a link, an attribute, a rule, a constraint or any other type of object. Only if an access type link from a task to an object exists, may that object be accessed with the operation specified by the access type link.

Deduced knowledge is derived from the triggering of rules. A rule can only be triggered, i.e., used to generate deduced information, if a user has access to the rule. A rule is triggered on that subset of the knowledge base which a user has permission to access. The deduced information can only be read. Access to deduced objects is possible by others, only as they are able to access and trigger a rule which creates the deduced object.

The GSM requires that a rule only be created by a task with owner access to the objects which are components of the predicate that specifies the rule. The reason for this requirement is that when the owner of a rule grants access to the rule, others may possibly discover the components of the rule. If the owner of the rule is also the owner of the objects used in the specification of the rule, then, since he is the owner, he has the right to let others discover these components.

Constraints are similar to rules since in their specification they may refer to various objects in the knowledge base. These components may become accessible to a task which receives access to the constraint. Therefore, only an owner of all components of a constraint can create a constraint in the first place. A constraint only applies to a user if the user has access to the constraint.

4 A RULE-BASED SECURITY ANALYSIS TOOLKIT

The relationship between the GSM and an implemented information system can be viewed as pictured in fig. 3. An access request to the information system goes to the GSM. If access permission for the current user and task is permitted, then data from the information system can be retrieved or modified according to the access type specified. However, since the GSM-based description of the relationships between users, tasks, and objects is a formal Telos knowledge base, the deductive querying and integrity checking facilities of a Telos implementation can be used for analyzing the information systems model in general at the class level, and the actual situation at the instance level of the model. In this section, we describe the rationale and implementation of such a toolkit we have developed for the Telos-based software information system, ConceptBase. As mentioned earlier, a more general

environment which covers the meta class level of basic policy specifications as well is still under development. We are working on this based along two threads: one is based on a mapping of GSM variants to SPM [Sandhu 88], the other on the use of model-generation algorithms such as SATCHMO [Bry et al. 88].

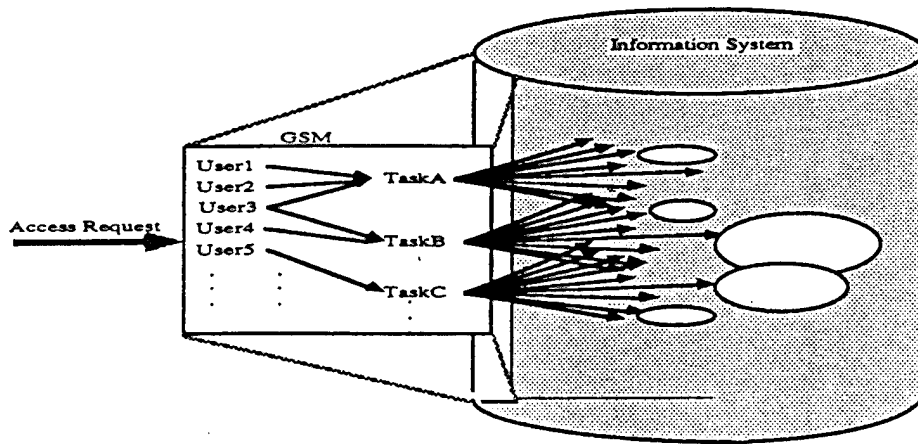


Fig. 3. Access request arrives via the GSM to the information system.

The potential applications of such a toolkit are manifold. Reasoning about security can be used to assess the impact of specific sets of knowledge classifications to determine appropriate security criteria. "Because its knowledge is encoded for logical processing, a knowledge base system is potentially able to reason about the closure of subsets of its knowledge base with respect to inference" [Garvey and Lunt 91, p.34]. Thereby one can find risky situations where insufficient security criteria have been specified for information, or costly situations where information has been over-classified.

Reasoning about security enables a user to look back and explain or justify prior actions in a system. Such a capability is particularly needed by an auditor who must be able to determine who was permitted to access information and why that permission existed.

Reasoning about security enables a preview of the potential impact of a user or an operation. What information can the user already access? To whom may the user pass on information? What new rights could a user possibly get by exercising the current rights? Is there a way a user can get access to a particular object?

Security and access control is required when integrating data from several organizations. This situation arises at the United States Environmental Protection Agency (EPA). A law, "The Toxic Substances Control Act", requires chemical manufacturers to submit details of their products to the EPA. The EPA is required to prevent the disclosure of this data since the information could be used to discover details about corporate strategies, pricing policies, research efforts and development plans. At the same time, "The Freedom of Information Act" requires non-sensitive information to be made public.

In the broader context of the four information systems engineering worlds mentioned in the introduction to this paper, the toolkit could also use reasoning about security to evaluate the privacy implications and requirements of the information in the knowledge base. What

permission must be granted from the subject of the information? How do the security criteria relate to the privacy requirements?

In our study, we have used the example of a hospital information system which requires a significant security component to ensure privacy and correctness of its information. Fig. 4 presents a subset of the information contained in a hospital system whereas fig. 5 shows a number of tasks in the hospital environment.

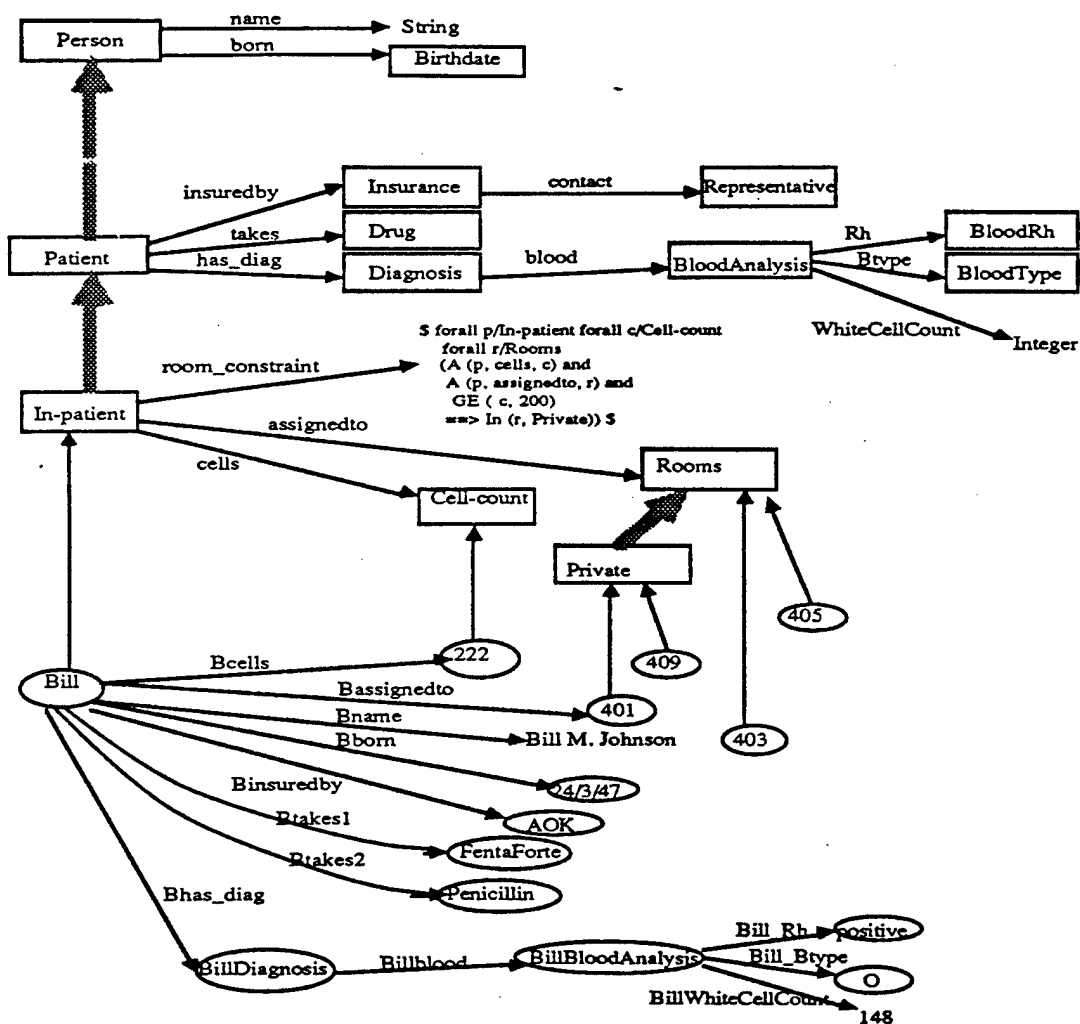


Fig. 4. Telos Hospital Example

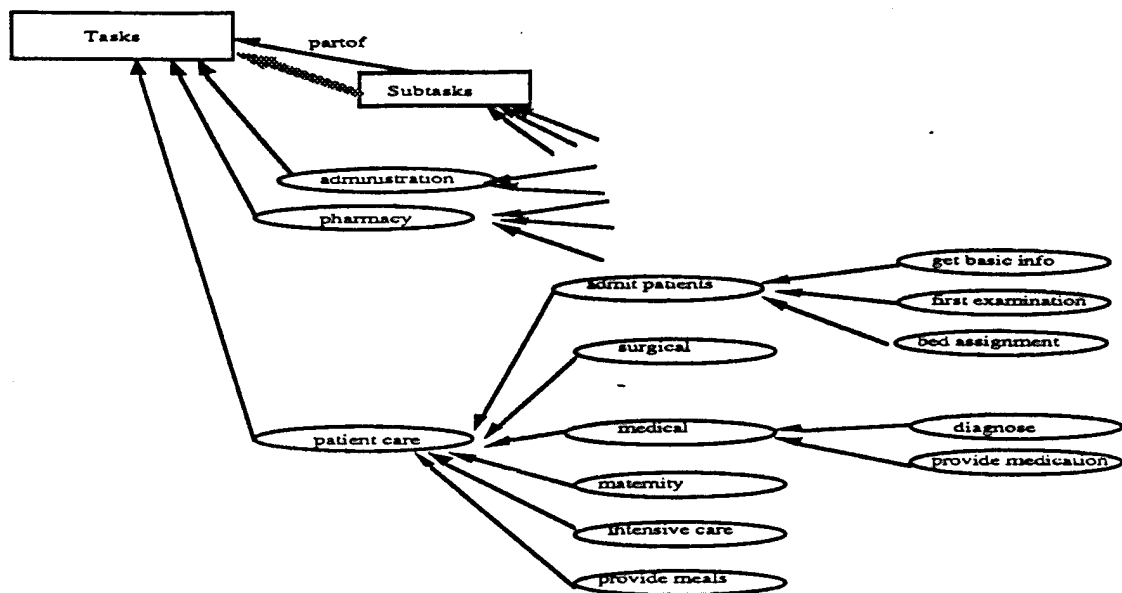


Fig. 5. Tasks in a Hospital environment.

4.1 Toolkit Overview

The interface of the security analysis tool is shown in fig. 6. Users are able to use the security analysis tool according to the nature and requirements of their task. A user of the security task may specify any class level object. That means that the security task may know the structure of the information and who may access information at the class level. Specific instances of information are only available to the owners of the related classes and the owners of the information itself.

Security Analysis Tool

☐ Current Access
☐ Potential Access
☐ Access as of _____

Object(s)

Accessible By:

Task	User	Access Type:

Explanation

Fig. 6. Interface of Security Analysis Tool

Information can be requested according to the historical, current or future status of the knowledge base. Historical information requires the specification of a date and will check the access restrictions of the system as they existed at that date. Current status concerns the system as it exists now, while the future status provides information as the system may exist at a future point in time if all users and tasks receive access to all information they are potentially capable of receiving.

The user indicates which object is to be examined. If more than one object is specified then the security analysis tool reports on those users and tasks who have access to all of the specified objects. For example, a security administrator may want to know which users and tasks may access both, a patient's insurance and a patient's diagnosis.

The response of the security analysis tool is shown in the bottom half of the interface. A list of all tasks and users who may access the specified objects is returned along with the access types. Thereby the user knows who had, has, or is potentially able to have access to the specified objects. A user's response could be to revoke access permission or to specify the denial of access to particular users or tasks.

Further information provided by the security tool enables a user to discover why access permission to an object is granted. The explanation component lists the objects in the knowledge base, including the rules, which result in a user and task receiving particular access permission.

4.2 Access and Potential Access to an Object

Determining who had access in the past, has access currently, or could have potential access to information in the future, is dependent on the security model which defines the capabilities of users in a system and the criteria necessary to access information. The GSM permits access to information if there is a link from the user's task to the object. These links may be specified explicitly or implicitly by application of the rules in the GSM. Potential access is determined by checking which users and task possess, or may be granted access to an object.

Explicit Access. Explicit access means that an access type link exists from a task to the object. The security analysis tool grants the leader of the task which owns an object permission to read the users, their roles, as well as the tasks and the access types which are associated with the owner's object. The owner of the object Bill may know all users and tasks who may access the object Bill.

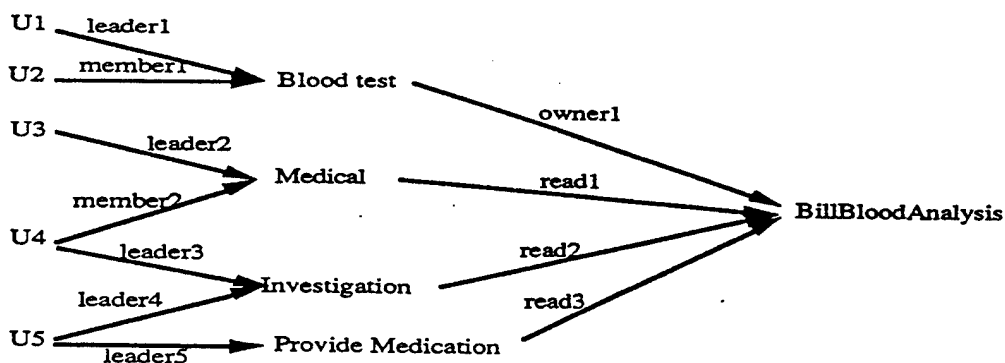


Fig. 7. Users and tasks with explicit access to an object.

Fig. 7 shows five users in four tasks with explicit access to the object BillBloodAnalysis. Since the Blood test task has owner access to the object, the leader of the Blood test task, U1, may know which other tasks and users may access BillBloodAnalysis, namely the users, their roles, their tasks and the access types which are shown.

In Telos these read access capabilities for the owner of an object are expressed as follows (notice that the information also requires that one is the leader of the task):

```
forall CU / currentuser, forall CT / currenttask,
forall X, ID1, ID2 / Proposition, forall T / tasks,
forall AT / tasks!accesstypes, forall R / users!roles,
forall U / users,

    A (CT, owner, X)
    and A (CU, leader, CT)
    and Prop (ID1, T, AT, X)
    and Prop (ID2, U, R, T)

==>  A (CT, read, ID1 )
      A (CT, read, T)
      A (CT, read, ID2 )
      A (CT, read, U)
```

The determination of who has access to deduced information is a more complex process. Deduced information is information that comes from the application of one or more rules. Deduced information does not have an owner. Even though deduced information is generated by an action of a user, it is not correct to speak of the user triggering a rule, as the owner of the information deduced from that rule. Ownership would include the capability of granting others access to the deduced object, as well as being able to delete the deduced object. These operations are not possible without modifying the rule from which the information was deduced.

Furthermore, the information generated by a rule depends on the knowledge base within which the rule is triggered. Therefore to determine who has access to some particular deduced information would require being able to trigger the rules accessible to each user in the subset of the knowledge base which each user can access. It is therefore necessary to ask who has access to the rule which possibly leads to the deduced information, rather than to request who has access to some deduced information.

Implicit Access. Implicit access comes from the application of rules defined in the GSM. There are three areas where implied access types are generated: implied read rule, owner of class object, readquery access type.

The implied read access rule states that access to a link implies access to the source and destination of the link. For example, in fig. 7, if there was an attribute object Patient!test which has as destination the object BillBloodAnalysis, then all those who could read object Patient!test also have implicit read access to object BillBloodAnalysis.

The GSM also provides implicit access to tasks which are the owners of a class object. Owner access to a class object implies access permission to objects "above" and "below" the object with the owner access in order to enable full inheritance capabilities. The owner of the object BloodAnalysis of which BillBloodAnalysis is an instance, would have implicit access to the object BillBloodAnalysis.

A readquery provides a task with access to all objects which satisfy the query, with the query being evaluated when access is requested. To determine which users and tasks have access to an object, one must also determine if the object satisfies a query to which readquery access has been granted. If in fig. 7, the object BillBloodAnalysis satisfies a query object X, then all tasks with a readquery access type link to object X, have implicit access to object BillBloodAnalysis.

Potential Access. Reasoning about security allows one to determine all potential tasks which may receive access to an object, as well as all users who are, or may be associated with these tasks. In the following we assume that the user component of the GSM is fixed, i.e., no more users or supervise links are added to the knowledge base. In addition we assume that no further transfer links are created. By these assumptions, an analysis of potential access becomes possible through deductive database technology. If these assumptions are too limiting then further rules must be set up to handle additional potential users and tasks (and a careful meta level study of policy definitions will be needed for acyclicity and attenuation).

We first consider *potential users* for a task. Potential users are users who are, or may be, associated with a task. We use the term potential leaders since all potential users of a task, are potential leaders of that task.

The leader of a task may assign to his task or to his subtasks, as members or leaders, those users with whom he has a supervises relationship. The supervises relationship is determined by a supervises link to another user or by the rule which states that the leader of a task supervises all users associated with the task.

The role potleader indicates a user who is a potential leader of a task and is expressed by the following two rules:

```
forall U1, U2 / users, forall T / tasks,
    ( A (U1, leader, T)
    or A (U1, potleader, T) )
    and A (U1, supervises, U2)
    ==> A (U2, potleader, T)

forall U / users, forall T1, T2 / tasks,
    ( A (U, leader, T1)
    or A (U, potleader, T1) )
    and A (T2, partof, T1)
    ==> A (U, potleader, T2)
```

Fig. 8 provides an example to show the range of potential users which can be assigned to a task. Currently only User1 and User2 have access objects associated with TaskA. As leader, User1 may assign additional users to TaskA, namely users whom he supervises, User10, User11, User100, User 101 and User110. If User1 should assign User2 also as leader of TaskA, then User2 would also have the capability to assign users to TaskA, namely User20, User21, User200, User210 and User211. Therefore all users shown are potential leaders of TaskA. All these users would also be potential leaders of a subtask of TaskA.

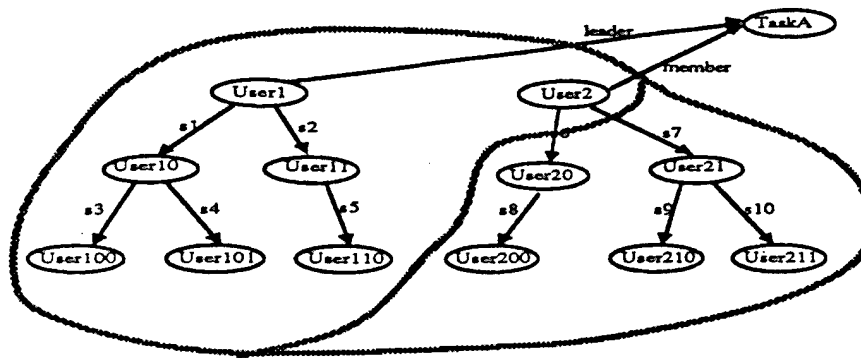


Fig. 8. Potential users who may be associated with a task.

Potential tasks w.r.t. an object, are tasks which may receive permission to access that object. A task receives access to an object in two ways. The first method is to be granted access permission from a task which owns the object. The second way that a task may receive access permission to an object is if it is copied from the parent task.

A task with owner access type for an object may grant access permission to those tasks to which a transfer link exists. Therefore all tasks to which a transfer link exists are potential owners of the object. Each potential owner may in turn pass the object on to those tasks it has transfer links to.

The following rule determines for an object X, all those tasks which are the destination of a transfer link from an owner, and therefore are potential owners of the object X.

```
forall T1, T2 / tasks, forall X / Proposition,
    ( A (T1, owner, X)
      or A (T1, potowner, X) )
    and A (T1, transfer, T2)
    ==> A (T2, potowner, X)
```

Fig. 9 shows a task T1 which is the owner of an object X. Task T1 has a transfer link to task T2, and task T2 has a transfer link to task T3. Since task T1 is the owner of object X, and has a transfer link to task T2, task T1 may grant owner access of object X to task T2. The dotted line marked potowner1 indicates that task T2 is a potential owner of X. Should T2 receive owner access to object X, then it may grant owner access to task T3, since a transfer link exists from T2 to T3. T3 is thereby a potential owner of X.

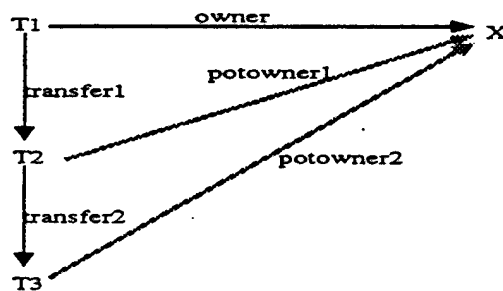


Fig. 9. Potential owners of an object

The second way a task may receive access permission to an object is from the parent task. In the GSM a parent task may copy the same access permission it has for an object, to its subtasks. Each subtask in turn, may copy the access permission it has to its subtasks. Therefore access permission to any task results in potentially all of its subtasks receiving the same access permission.

The fact that access permission associated with a task can be copied to its subtasks is expressed by the following rule:

```
forall T1, T2 / tasks, forall X / Proposition,
forall AT / accesstypes,

    A (T1, AT, X)
    and A (T2, partof, T1)
    ==> A (T2, potaccess, X)
```

We use potaccess to indicate any type of access. The actual potential access type is that possessed by the parent task.

In order to determine potential users and tasks we have defined users who may be associated with a task by the potleader role, and tasks which may be associated with an object with the potowner and potaccess access types. The owner of an object is granted access permission to all potential users and tasks of his object.

4.3 Explanation Component

The explanation component of the security analysis toolkit allows a user to determine how or why access to an object is granted. Access permission to an object may exist because of an explicit access type link between a task and the object. Access permission to an object may also exist from the application of rules which provide an implicit access type link between a task and an object. The explanation component displays the access type links and rules which result in the access permission.

The explanation component creates a query which searches the knowledge base for a particular task, access type and object. All rules and objects which contribute to the successful discovery are provided in response to the query. The explanation component may also be used to explain how potential users and tasks may receive access to an object. This may be used in order that such potential access can be prevented before it exists.

5. CONCLUSION

Our goal in introducing task as a major concept in IS security modeling has been to make security impact analysis fit with the usual IS engineering process. A second potential benefit is a more fine-grained approach to the security analysis itself: there is an additional degree of freedom in defining user groups due to the assignment of task combinations to these groups.

The security toolkit, as implemented so far, has the advantage that no further software had to be written beyond the standard deductive database optimization techniques anyway present in our system. (There has been a substantial implementation effort, though, for providing an efficient way to store access rights, see [Steinke 92]).

On the other hand, we are quite aware that this paper is just a first step. In particular, we have mentioned the need for characterising GSM meta level policies in a way similar to those in SPM or related models. We do not see major problems as long as we follow simple policies such as providing each task leader full information about all the subtasks he creates (attenuation). However, it remains to be studied whether such a policy is compatible with the delegation and privacy principles in organizations, and what we can say should that not be the case.

Literature

- [Baldwin 87] Baldwin, R.W., Rule Based Analysis of Computer Security, In *Proc. of IEEE Compcon 87*, Feb. 87, 227-233.
- [Berson & Lunt 87] Berson, T.A., Lunt, T.F., Multilevel Security for Knowledge-Based Systems, *Proc. of the 1988 IEEE Symposium on Security and Privacy*, 235-242.
- [Borgida et al. 87] Borgida, A., Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y., The Software Development Environment as a Knowledge Base Management System, In Schmidt/Thanos (eds.) *Foundations of Knowledge Base Management*. Springer-Verlag, 411-440.
- [Bry et al. 88] Bry, F., Decker, H., Manthey, R., Integrity constraint satisfaction and satisfiability in deductive databases. *Proc. EDBT 88*, Venice.
- [Garvey & Lunt 91] Garvey, T.D., Lunt, T.F., Multilevel Security for Knowledge Based Systems, Tech. Report SRI-CSL-91-01, SRI International, 1991.
- [Glasgow et al. 91] Glasgow, J., MacEwen, G., Panangaden, P., A Logic for Reasoning about Security, unpublished paper, Aug. 7, 1991.
- [Harrison et al. 76] Harrison, M.A., Ruzzo, W.L., Ullman, J.D., Protection in operating systems, *Communications of the ACM*, 19, 8, 1976, 461-471.
- [Jarke 90] Jarke, M., DAIDA - Conceptual Modeling and Knowledge- based Support of Information Systems Development Processes, *Technique et Science Informatiques*, Vol. 9, No. 2, 121-133.
- [Jeusfeld & Jarke 91] Jeusfeld, M., Jarke, M., From relational to object-oriented integrity simplification, *Proc. 2nd Intl. Conf. Deductive and Object-oriented Databases*, Munich 1991.
- [Jeusfeld & Staudt 91] Jeusfeld, M., Staudt, M., Query optimization in deductive object bases. In Freytag/Maier/Vossen (eds.): *Query Processing in Object Databases*, Morgan Kaufmann, to appear.
- [Mylopoulos et al. 90] Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M., Telos: Representing Knowledge about Information Systems, *ACM Trans. on Information Systems*, Vol 8, No. 4, Oct. 90, 325-362.
-]Rose et al. 92] Rose, T., Maltzahn, C., Jarke, M., Integrating object and agent worlds, *Proc. 4th Intl. Conf. Information Systems Engineering*, Manchester, UK, 1992.
- [Sandhu 88] Sandhu, R.S, The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes, *Journal of the ACM*, Vol. 35, No. 2, 1988, 404-432.
- [Steinke 91] Steinke, G., Towards a Strategy for Achieving Security and Multi-User Integrity in Knowledge Base Systems, *Proc. 2nd Intl Workshop on the Deductive Approach to Informaiton Systems and Databases*, Costa Brava, Catalonia, Spain, Sept 1991.
- [Steinke 92] Steinke, G., Task-Based Security for Knowledge Base Systems, PhD Dissertation, University of Passau, 1992.

Toward a Tool to Detect and Eliminate Inference Problems In the Design of Multilevel Databases*

Thomas D. Garvey[†] Teresa F. Lunt[‡] Xiaolei Qian[‡] Mark E. Stickel[†]
Artificial Intelligence Center[†] and Computer Science Laboratory[‡]
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

Abstract

We are developing a new tool, DISSECT, for detecting and removing specific types of inference problems in a multilevel database system. This tool could be used by a data designer to analyze a candidate database schema for potential inference problems. DISSECT will provide facilities for characterizing database schemas in terms of their explicit and implicit relations, constraints, and rules; for recognizing paths from low data to high information through relations, constraints, or rules; for suggesting methods for eliminating inference paths; and for recognizing inference channels leading to partial inferences. The tool will allow the graphical representation of the structure of a multilevel database, the interactive detection of specific types of potential inference channels in the database, and the removal of those channels. In its initial form, DISSECT will facilitate experimentation with different methods for analyzing databases for inference problems.

1 Introduction

The *inference problem* in a multilevel database arises when a user with a low clearance, accessing information of low classification, is able to draw conclusions about information at higher classifications. There is often no immediate or obvious connection between the low data and the inferred high data; however, an inferential chain may link the low and high data through low relations, constraints, and rules, some of which may not be explicitly

*This research was supported at SRI by the U. S. Air Force, Rome Laboratory and the U. S. Department of Defense, Advanced Research Projects Agency, under contract F30602-91-C-0092.

stored in the database. An inferential link of this nature which may allow information to flow from a high security class to a low security class is termed an *inference channel*.

The advent of operational multilevel database systems brings the capability for enforcing mandatory security policies that prohibit the unauthorized disclosure of information to uncleared or insufficiently cleared individuals. However, multilevel database systems do not provide effective mechanisms for preventing an unauthorized person from inferring high information from low data. This problem is especially serious when information to which the unauthorized user might have legitimate access (or that might even be common knowledge) forms part of the inferential chain. The problem of characterizing, detecting, eliminating, and alleviating such inference channels is a different problem that has received relatively little formal attention and may ultimately be of much greater significance than the access-control problem. The inference problem is especially difficult when it often involves information that is not explicitly represented in the database and thus not easily considered by automated tools. In characterizing inference problems and in developing methods for their detection, this implicit aspect must be made explicit.

Some of the difficulties with identifying inference channels arise from the following issues.

- The channels may involve lengthy chains of inference.
- Certain knowledge about the database may exist as implicit rules or constraints embedded in programs, and may not be explicitly available as part of the database definition.
- Even though sensitive information may not be directly inferable, it may be *partially inferable* in that unclassified information may allow a user to reduce the set of possible values that can be assigned to a classified datum.
- A low user may have access to externally available information, which, when combined with the information to which he has legitimate access, may allow the inference of high information.
- The security manager may have improperly classified information that is generally known by unclassified users believing that other classified information is thereby protected.

Preventing all inference of sensitive information by uncleared individuals is a problem of overwhelming difficulty. Its solution requires, in principle, a complete model of all knowledge and information that might be used to infer the sensitive data, which is generally impractical, as well as the ability to recognize all sensitive implications of that information, which is generally impossible.

In this paper we describe our ongoing work to improve the security of a multilevel database by providing tools for inference analysis and control. Our approach is to represent explicitly much of the information that might normally be used to infer sensitive information. In particular, we intend to represent constraints and rules that often express implicit extensions to a database system. In many cases this information will be obtained by

querying the data designer regarding general knowledge that might be used to infer sensitive data and might be available externally. Furthermore, we shall interactively elicit from the data designer specific information that might be used to infer particularly sensitive data, in order to ensure that it is adequately protected. When inadequately protected data is recognized, the tools will suggest modifications to the data design to provide the necessary protection.

A particularly difficult and important problem involves the ability to draw partial conclusions about sensitive information, even when the information may not be directly inferable. This problem argues for a more quantitative model of inferability. We will address this problem with an uncertainty model of inference.

2 A Tool for Inference Control

Among the difficult challenges to be faced by the first designers of applications using multi-level database systems is that of labeling the data elements and tuples in such a way that

- the labels accurately reflect the real-world classification of the information, and
- the labels adequately protect the information from inference.

The first of these two aims is often difficult, but in general it is expected that the task of assigning labels to data that accurately reflect the real-world classification of the information will be within the grasp of the data designer. However, the second of these two aims, that of ensuring that the assigned labels adequately protect the information from inference, will be much more difficult, if not impossible, for the human data designer to attain. An automated tool that can identify potential inference channels would contribute greatly to the overall assurance that the data was secure (as has been noted, for example, by Buczkowski [1] and Lin [7]). We describe here our method for developing such a tool, called DISSECT (Database Inference System Security Tool).

It has been shown that many inference problems can be avoided by proper data design [8]. Thus it should be possible to design a tool to detect such problems at data design (or redesign) time. Such an approach will not detect the additional inference problems that may depend on the actual values of the data rather than depending simply on the design of the multilevel data structures (we discuss this issue further in Section 4). However, this approach is appropriate for three reasons. First, it promises to address a large portion of the inference problem. Second, it is a much simpler problem to address the problems that arise at data design time. Third, the analysis need only be performed at data design (or redesign) time, a relatively infrequent occurrence. To address the problems that depend on the values of the data, not only is the scope of the analysis much larger, but it must be redone each time the database changes.

2.1 Use of the Tool

We envision that our tool will be used as follows. The data designer responsible for designing the multilevel relations that support an application, will develop a candidate design encoded in a set of structures called the *database schema*. The database schema contains a listing of the multilevel relations in the database and includes information such as the table names and classifications, the names, types, and classification ranges of the columns in the tables, the formulas for the integrity constraints that have been defined, primary and foreign key designations, and other such information. The data designer defines these data structures to the database system using the data definition language provided as part of the database system. For example, a multilevel relation FLIGHT can be defined as follows.

```
create table flight
  (group(date-time-group char, dest char) [U] primary-key,
  flightno number [U] foreign-key mission,
  aircraft char [U:S])
```

We see that the table FLIGHT has attributes (column names) DATE-TIME-GROUP, DEST, FLIGHTNO, and AIRCRAFT, where the uniformly classified group (DATE-TIME-GROUP, DEST) is the primary key for the relation and FLIGHTNO is a foreign key to another multilevel relation MISSION. (We say that a group of attributes is *uniformly classified* if, within every tuple of the multilevel relation, they share a common label.) We also see that the values for the attributes DATE-TIME-GROUP, DEST, and FLIGHTNO must be UNCLASSIFIED, whereas values for the attribute AIRCRAFT may have access classes ranging from UNCLASSIFIED through SECRET. The access class of the table name (and of the schema information itself) is the same as that of the subject that created the table. In this case, the table must have been created by an UNCLASSIFIED subject, since we require that the subject class be dominated by the lower bound of all the attribute access class ranges.

Once these table definitions (and related integrity constraints) have been declared, the data designer will use DISSECT to analyze the candidate design for potential inference problems. (Such analysis may also be done incrementally as the data designer first builds a subset of the tables and then adds to this collection.) DISSECT will take the database schema as input and will display the data design graphically on the screen in a manner similar to Figure 1.

However, it is not sufficient merely to show that none of the high information that is to be stored in the database can be inferred from the low data that is to be stored in the database. This is because part of the information needed to make the inference will often *not* be stored in the database, but may be general knowledge about the application. For example, suppose we want to prevent users from inferring the sensitive relationship between a flight's departure time and destination and its mission. Suppose the data designer created the following set of relations.

```
FLIGHT[U]((DATE-TIME-GROUP, DEST)[U], FLIGHTNO[U], AIRCRAFT[U:S])
MISSION[U](FLIGHTNO[U], TYPE[U])
CARGO[U](FLIGHTNO[U], CARGO[U])
```

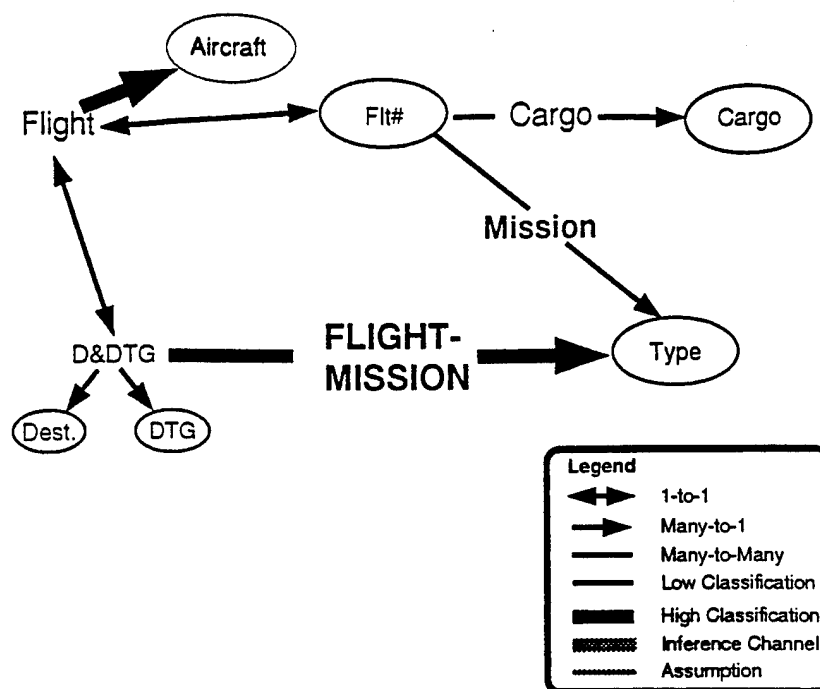


Figure 1: Graphical Data Design

The sensitive relationship is easily computed by joining the two relations on flight number. Figure 2 shows the inference path as it would be highlighted on DISSECT's graphical display.

One way to fix the problem is to upgrade a node or link in the graph. Suppose the data designer upgrades the MISSION table to SECRET, as follows.

```
FLIGHT[U]((DATE-TIME-GROUP, DEST)[U], FLIGHTNO[U], AIRCRAFT[U:S])
MISSION[S](FLIGHTNO[S], TYPE[S])
CARGO[U](FLIGHTNO[U], CARGO[U])
```

This data design seems safe, since it prevents the deduction. However, if a smart user knows a plane's cargo, it may be possible to guess the flight's mission. That is, a flight's mission may be inferred (or partially inferred) from its cargo. This means that the sensitive relationship between a flight's mission and its departure time and destination may still be compromised. Such "near keys," or attributes that allow partial inference of identifying information, have been called *identificates* [15]. The fact that the cargo allows a partial inference of the mission is *not explicitly stored in the database*. Thus, in order to do an adequate inference analysis, additional information must be elicited from the data designer. This particular connection is one that the data designer, or someone familiar with the application, should know. Thus, once it receives the database schema as input, DISSECT will query the data designer for such relationships, and will add these to its inference graph. Figure 3 shows the augmented inference graph.

This example illustrates an important fact: databases represent a fragment of reality, but connections between the database and other nonrepresented facts and relations are real and form the context for its interpretation. This context must be accounted for in evaluating a database for inference problems. In all of the other work to date on the inference problem, the researchers have confined themselves to simply uncovering which high *stored* information

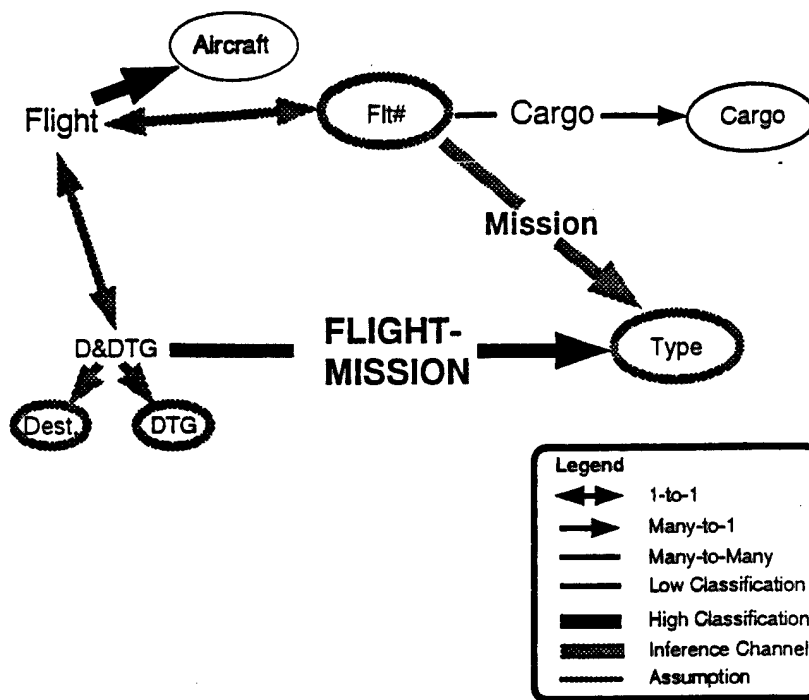


Figure 2: Inference Path

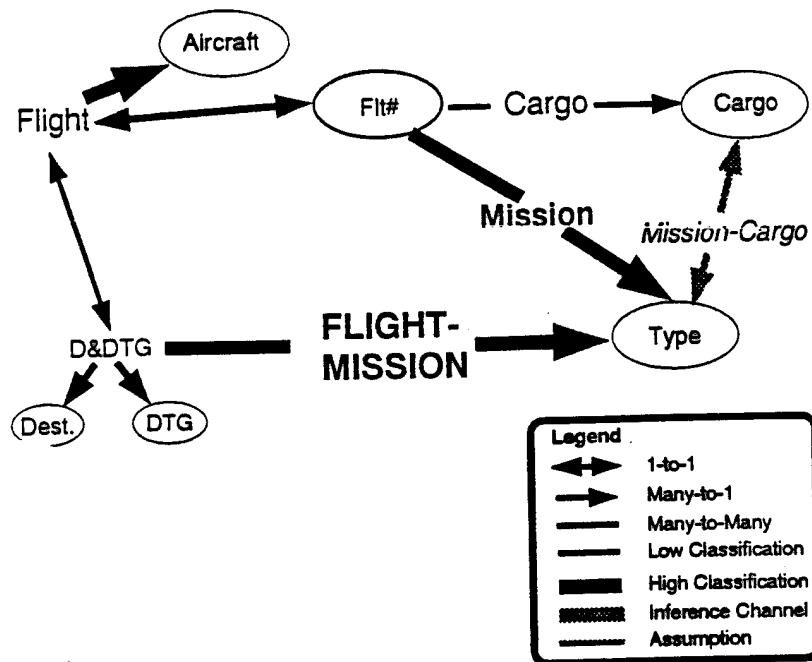


Figure 3: Augmented Inference Graph

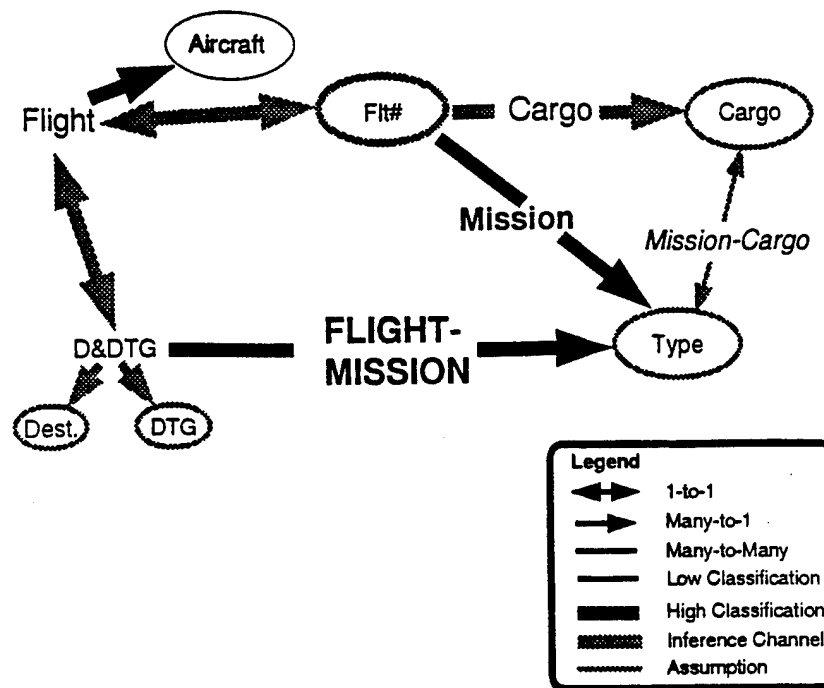


Figure 4: Inference Path

can be directly deduced from which low *stored* information (e.g., [17, 19, 6, 10]). As this example illustrates, such an approach is insufficient to address the problem meaningfully.

In the above example, once this additional information has been obtained from the data designer, DISSECT would highlight the inference path on the graph, as shown in Figure 4. DISSECT could suggest a number of redesigns to remove the problem. This example highlights the need to know which attributes act as partial keys and to understand how these contribute to making inferences.

When DISSECT discovers an inference chain, it may be possible to remove the problem through raising the classification of some critical node or arc (attribute or table) on the graph, or through splitting relations to remove or upgrade associations among attributes. Once the changes have been made, the data designer would re-run DISSECT to ensure that no new inference problems have been introduced by the restructuring. In some cases it may not be practical to upgrade some critical piece of information; in this case a cover story may be appropriate [4].

DISSECT can be used iteratively by the data designer as new relations or constraints are added to the database. DISSECT will check for new inference channels, query the user for new inference rules, and recommend steps to eliminate any that are uncovered.

We do not expect to be able to recognize and eliminate *all* inference channels. However, DISSECT should assist in removing many of the most serious ones.

The internal logic used by DISSECT is inspired by the Nonmonotonic Typed Multilevel Logic developed by Thuraisingham [18] and is discussed in Section 3.2. This section also discusses the mapping between the graphical representation and the multilevel logic. We have developed automated techniques for following inference chains through the semantic model in order to recognize inference channels from low data to high data. Our reasoning procedures

for detecting connections between low data and high data are discussed in Section 3.3. The mapping between the multilevel database schema and the graphical representation is discussed in Section 3.1.

Although our work to date has emphasized the development of an interactive tool to be used for analyzing a database for potential inference channels and for assisting in the elimination of such channels, as a secondary goal, we seek to develop a quantified theory of security based on the probability that high information may be inferred from low information. Our approach synthesizes aspects of information-theoretic approaches and probability techniques and is discussed in Section 4.1.

3 Formalizations

DISSECT incorporates three major components for every multilevel relational database: a multilevel relational schema, a semantic net for the graphical representation of schemas and inference channels, and a multilevel logic for reasoning with schemas and inference channels. Since DISSECT detects potential inference channels with logical reasoning, and the data designer interacts with the semantic net to eliminate inference channels when they are detected, we provide algorithms to map the semantic net to and from the multilevel schema as well as the multilevel logic.

3.1 The Semantic Net

To address the inference problem, it is necessary to define explicit, security-relevant database semantics. A convenient representation for both data semantics and *secrecy semantics* [15] is the *semantic net*. With this approach we represent a multilevel database as a graph structure explicitly describing the semantic relationships between nodes that represent concepts. This provides a clear and understandable overview of the connections among data and, in particular, what aspects of the data and relationships are classified, as shown in Figures 1, 2, 3, and 4. We are extending the semantic language to incorporate constraint relations, classification rules and other relevant knowledge directly as graphical links between appropriate nodes in the semantic graph.

Semantic nets are composed of nodes and arcs. Arcs represent binary relationships, and nodes represent concepts and n -ary relationships where $n > 2$. Each node or arc has an identifier and a value, which can be classified at various levels. As in [15] we use highlighting to represent classified nodes and arcs. For the example graph in Figure 1, AIRCRAFT represents a concept, MISSION represents a binary relationship between FLIGHT# and TYPE, and FLIGHT represents a ternary relationship. The identifier of AIRCRAFT is UNCLASSIFIED, while its value is classified ranging from UNCLASSIFIED to SECRET.

Every relation $R[X, K]$ has a name R , a set of attributes X , a primary key $K \subseteq X$, and zero or more foreign keys. Every attribute in turn has a name and a set of values. They can be classified at various levels. For relation $R[X, K]$ with n attributes A_1, \dots, A_n ,

$R[L](A_1[L_1], \dots, A_n[L_n])$ indicates that the relation and attribute names are classified at L , and the attribute values are classified at L_1, \dots, L_n respectively.

Assignment of classification levels to constructs in the semantic net is consistent if values are classified at least as high as identifiers, and if relationships are classified at least as high as arguments. For example, the identifier of TYPE could be UNCLASSIFIED, while its value could be classified as TOP-SECRET, in which case the binary relationship MISSION must be classified at least as TOP-SECRET.

Assignment of classification levels to components of multilevel relations is valid if attribute values are classified at least as high as relation and attribute names, and if entity integrity as well as referential integrity[9] are preserved. The three multilevel relations in the previous section are all valid.

The mapping of valid multilevel relations to semantic nets guarantees the consistency of the resulting graphs. The mapping consists of two steps: a concept construction step and a relationship construction step. In the rest of the section, we illustrate the mapping with the following multilevel relations, where the first attribute is the primary key except for the last relation, whose primary key consists of two attributes. The result of the mapping is shown in Figure 5.

```
FLIGHT[U](FLIGHTNO[U], (DATE-TIME-GROUP, DEST)[U], AIRCRAFT[U:S])
MISSION[U](FLIGHTNO[U], TYPE[S])
MISSION-TYPE[U](TYPE[U])
CARGO-TYPE[U](CARGO-NAME[U], UNIT-WEIGHT[U])
CARGO[U](FLIGHTNO[U], CARGO-NAME[C])
```

Every attribute A is mapped to a concept node N_A . If there is a unary multilevel relation $R_A[L](A)$, then the identifier and value of N_A are classified at L . Otherwise they are not classified. For example, attribute TYPE is mapped to a concept TYPE. Since there is a unary relation MISSION-TYPE[U](TYPE), the identifier and value of TYPE are UNCLASSIFIED.

Multilevel relations with the same primary key are grouped together. If there is more than one primary key attribute, then an n -ary relationship node is created to represent the primary key. The identifier of the primary key node is classified at the greatest lower bound of levels of relation names in the group. The value of the primary key node is classified at the greatest lower bound of levels of the primary key of relations in the group. For example, since the primary key of relations FLIGHT and MISSION are FLIGHTNO, relations FLIGHT and MISSION are grouped together. Because the primary key consists of a single attribute, we don't need to create a primary key node.

Attributes in the same group that are not primary key attributes are mapped to many-one binary relationships from the primary key node to the concept nodes that represent the attributes. The identifier and value of the binary relationships are classified at the levels of the relation name and attributes respectively. For example, attribute AIRCRAFT is mapped to a many-one binary relationship from FLIGHTNO to AIRCRAFT, whose identifier and value are UNCLASSIFIED and from UNCLASSIFIED to SECRET respectively. Likewise, attribute TYPE is mapped to a many-one binary relationship from FLIGHTNO to TYPE.

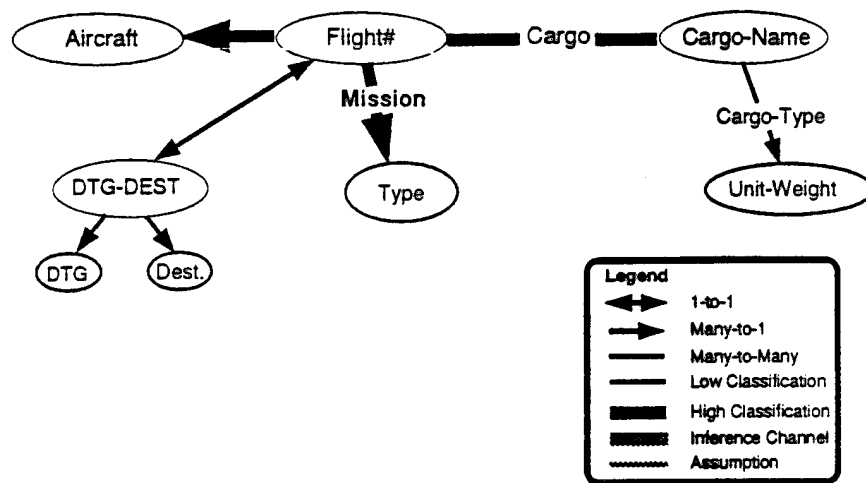


Figure 5: Multilevel Relations to Semantic Net

If the attributes of a relation R_1 consist of the primary key of another relation R_2 plus an attribute A , then R_1 is mapped to a many-many binary relationship between the concept nodes that represent A and R_2 . The identifier and value of the binary relationship are classified at the levels of R_1 and A respectively. For example, relation CARGO is mapped to a many-many binary relationship between CARGO-NAME and FLIGHTNO. The identifier and value of the binary relationship are classified both at CONFIDENTIAL.

If the attributes of a relation R consist of the primary keys of relations R_1 and R_2 , then R is mapped to a many-many binary relationship between the nodes that represent R_1 and R_2 . The identifier of the binary relationship is classified at the level of R , while its value is classified at the least upper bound of attributes in R . For example, relation CARGO is mapped to a many-many binary relationship between FLIGHTNO and CARGO-NAME, whose identifier and value are both classified at CONFIDENTIAL.

3.2 A Multilevel Logic

We use a first-order predicate calculus (FOPC) theory to represent the content and constraints of a database—all are encoded as nonlogical axioms of the theory. The function-free subset of FOPC appears adequate (cf. Datalog). We include the equality relation so that it can be used to express integrity constraints.

Consider the relation

MISSION(FLIGHTNO,TYPE)

with instance

MISSION(2,WAR)

which we might represent logically by the axiom

mission(2, *war*).

Suppose "The mission of flight 2 is war" is SECRET. Protection of the secrecy of this fact can be achieved in the database in several ways. For example, if

1. the mission relation is SECRET.
2. WAR is a SECRET mission-type.
3. the individual fact of flight 2's mission being WAR is SECRET.

Our graphical notation can represent these distinctions between classifying the presence of missions in the database, classifying a mission-type, and classifying the association between a flight and its mission.

These distinctions can also be mapped into a logical representation like Thuraisingham's Nonmonotonic Typed Multilevel Logic (NTML) [18] that attaches security levels to symbols as well as formulas. To every symbol is attached a security level called its inherent security level. The inherent security level of a formula is the least upper bound of the inherent security level of all the symbols appearing in it. The first two cases can be represented by making the inherent security level of certain symbols SECRET:

1. *mission*(2, *war*), where the security level of *mission* is SECRET
2. *mission*(2, *war*), where the security level of *war* is SECRET

The third case is handled by explicitly attaching a security level (greater than or equal to its inherent security level) to a formula:

3. *mission*(2, *war*):SECRET

A multilevel database can be regarded logically as a collection of databases: the database as seen by the UNCLASSIFIED user, the one seen by the SECRET user, and so on. Rather than try to represent all these views within one logical representation as NTML does, we adopt the conceptually simpler approach of representing the database as seen by the UNCLASSIFIED user by one logical theory, the one seen by the SECRET user by another, and so on. Each of these logical theories has a vocabulary—the set of symbols from which formulas are constructed. A symbol whose inherent security level in NTML is SECRET would be a member of the vocabulary of the SECRET theory and theories with higher classification and not of theories with lower classification.

The inherent security levels of symbols induce a hierarchy of first-order languages. The language—the set of legal strings of symbols—at the SECRET level contains the set at the UNCLASSIFIED level and is contained in the set at the TOP SECRET level.

The secrecy of *mission*(2, *war*) is achieved in cases 1 and 2 by declaring certain symbols SECRET rendering the formula inexpressible in the UNCLASSIFIED theory. If the symbol *mission* is SECRET, then the UNCLASSIFIED theory can contain no occurrence of it—*mission*(2, *war*) could be a legal formula in the SECRET theory but a syntax error in the

UNCLASSIFIED theory. If, on the other hand, the symbol *war* is SECRET, then missions can be discussed in the UNCLASSIFIED theory, but the symbol *war* cannot be used.

When, as in case 3, we want to classify an association between a flight and its mission instead of the mission relation or a mission-type, the SECRET and UNCLASSIFIED theories may have the same vocabularies but different nonlogical axioms: *mission*, 2, and *war* could all be legal symbols of the UNCLASSIFIED theory (and hence all theories) but the formula *mission*(2, *war*) could be an axiom of the SECRET theory but not the UNCLASSIFIED one.

Note that although the vocabulary of a theory always contains the vocabulary of lower classified theories and is contained by the vocabulary of higher classified theories, such inclusion is not necessary for the nonlogical axioms of the theories. In particular, polyinstantiated multilevel databases require that higher classified theories not always include all the nonlogical axioms of lower classified theories. For example, to represent a polyinstantiated database in which flight 2's mission is PEACE at the UNCLASSIFIED level and WAR at the SECRET level, the UNCLASSIFIED theory would include the axiom *mission*(2, *peace*) while the SECRET theory would include the axiom *mission*(2, *war*) instead. Thus, an axiom of the UNCLASSIFIED theory would be absent from the SECRET theory.

There is considerable conceptual simplification in representing a multilevel database as a set of theories instead of a single multilevel theory as in NTML. The "nonmonotonic" aspect of NTML is invoked whenever some formula is true at some security level and false at a higher security level. The deduction rule in NTML needed to accommodate this is the Deduction Across Security Levels (DASL) rule. The DASL rule states that a formula is true at some security level if it is not contradicted at that level and is true at the next lower security level, i.e., the default is to inherit a formula's truth value across security levels.

The DASL rule is complicated, potentially inefficient (determining whether a possibly inheritable fact is contradicted at the next level may even be undecidable), and not always coherent (If *P* and *Q* are facts at the UNCLASSIFIED level and $\neg(P \wedge Q)$ a new fact at the SECRET level, which of *P* and *Q* is not inherited from UNCLASSIFIED level to SECRET level to avoid contradiction?). We avoid the complexity of nonmonotonic reasoning [5] by our decision to represent data at different security levels by different explicit sets of nonlogical axioms instead of using a rule of inference to propagate data from one level to the next. Consideration of which data to inherit from one level to the next is done when the set of theories is created instead of every time a proof is attempted in a single multilevel theory. It is preferable that our multilevel logic be the result of a nonmonotonic reasoning process instead of an input to one. Moreover, as we often regard the multilevel logical representation of a database as a theoretical construct that we reason about abstractly, we may never have to actually do any nonmonotonic reasoning, since we will not actually construct a set of theories from the specific data instances of a real database.

Like NTML, our logical representation is typed or many-sorted. The domain is divided into many nonempty sorts such as flight-number, mission-type, etc. Constants and variables are declared to be of certain sorts: 2 is of sort flight-number, WAR and PEACE are of sort mission-type. Constants may be of more than one sort. For example, *Acme* might be of sort customer and supplier. *Acme*'s status as a customer can be kept SECRET while its

status as a supplier can be UNCLASSIFIED by declaring *Acme* to be of sort supplier in the UNCLASSIFIED theory and of sorts customer and supplier in the SECRET theory. Predicate arguments are required to be of specified sorts. For example, the *mission* predicate's first argument must be of sort flight-number and its second argument must be of sort mission-type to be syntactically valid. Besides the other notational and efficiency advantages of many-sorted logic, many-sortedness provides the vital feature of allowing symbols to be legally used in some parts of the database and not others. An important feature of many-sorted logic is that if the axioms are well-sorted (syntactically legal with respect to sorts) then their consequences will also be well-sorted. We will assume a multilevel database whose operations can be modeled by such well-sortedness preserving inference.

3.3 Inference Analysis

Most inferential security problems fall into one of three distinct classes, based on the degree to which high data may be inferred from low data. A *deductive channel* is the most restrictive type and occurs when a formal deductive proof of the high data can be derived from the low data that is directly available in the database. An *abductive channel* exists when a deductive proof from data explicitly in the database or implicitly derivable from data in the database may not be possible, but where a deductive proof *could* be completed by making assumptions about certain low data. In this case, an abductive proof is possible (*abduction* is a non-valid inference process where one reasons from an observation to a possible cause for the observation). The third type of channel is the *probabilistic channel*. It exists when it is possible to estimate the likelihood of the truth of a high data element based on likelihoods known or computable for low data. Probabilistic channels are discussed further in Section 4.1.

Our multilevel logic provides a simple characterization of deductive inference channels. The database is modeled by a set of theories. A deductive inference channel exists if an axiom of a high theory that is not also an axiom of a lower theory is nevertheless deducible from the axioms of the lower theory. Note that although security rules stipulate that data derived from high data be classified high, such derived data may be deducible from low data as well without there being any security flaw. For example, that the company president's salary is \$2,000,000 may be SECRET, but the derived data that it exceeds \$50,000 (a logical consequence of its being \$2,000,000) may also be reasonably concluded from UNCLASSIFIED information with no security violation.

In general, many formulas in the deductive closure of the high classification theory are also in the deductive closure of the lower classification theory. When the axioms of the high theory include those of the lower one (e.g., no polyinstantiation), *all* formulas in the deductive closure of the lower theory are contained in the deductive closure of the high one. To distinguish between safe and unsafe consequences of the low data, we declare a security violation only if an *axiom* of the high theory, i.e., explicitly protected information in the database, is derivable from the low data.

Figure 2 shows an easily discovered deductive channel. By the database definition, the TYPE entity is accessible from both the low MISSION and high FLIGHT-MISSION relations.

Thus, we have the axiom

$$flight(dest, dtg, fltno, ac) \wedge mission(fltno, type) \supset flight-mission(fltno, type).$$

Assuming the database is populated, i.e., that

$$(\exists dest, dtg, fltno, ac) flight(dest, dtg, fltno, ac) \wedge mission(fltno, type)$$

an instance of *flight-mission* is easily shown to be deducible thus demonstrating a deductive-channel security flaw.

As mentioned before, this flaw can be eliminated by upgrading the MISSION table to SECRET. The deductive channel is blocked. Nevertheless, a problem, in the form of what we call an abductive channel, remains. In general, we expect deductive channels to be relatively rare and easily discoverable security flaws; more common and hard to find are abductive and probabilistic channels.

Abductive reasoning is a distinctly different form of reasoning from deductive reasoning and is not limited to demonstrating that a formula is a consequence of a theory. In abductive reasoning, the objective is to find assumptions that will allow an otherwise unprovable formula to be proved from a theory. Classical applications of abduction include attempting to find explanations for observations, e.g., to explain the observation that the grass is wet, we might hypothesize that it rained or that the sprinkler was on. Assuming either of these would allow us to conclude logically that the grass would be wet and thus are possible explanations for that observation. Operationally, abductive reasoning is handled similarly to deductive reasoning. While deductive reasoning may backward chain via implications from the conclusion to prove to subgoals to prove, grounding this process with facts, abductive reasoning may similarly backward chain via implications, but with the added provision that some of the subgoals may be assumed instead of proved to complete an abductive explanation.

The vocabulary of possible assumptions and the reasoning by which they can be used to explain or prove something cannot be generated from thin air, but must be elicited from the data designer. In the example of Figure 4, the data designer indicates that a flight's cargo can sometimes be used to predict its mission. This might be encoded by the rule

$$cargo-predicts-mission(x, y) \wedge cargo(fltno, x) \supset mission(fltno, y),$$

which states that in certain cases, formalized as those situations in which the formula *cargo-predicts-mission(x, y)* is true, knowing the cargo of a flight will lead to knowing its mission.

An abductive inference channel can be shown to exist by finding an abductive proof that high facts of either the originally protected FLIGHT-MISSION table or the newly upgraded MISSION table can be concluded from the theory of database plus the instances of the assumption *cargo-predicts-mission(x, y)*.

3.4 Removing Discovered Problems

Once an inference channel has been identified, it must be eliminated by somehow breaking the logical chain from low data to high data. The choice of the best point to break the

chain may be determined by several factors. In principle, it is desirable to upgrade as little information as possible (each newly classified datum must itself be checked for new inference channels).

DISSECT will use heuristics such as the following.

1. Sensitive associations among entities of different types—that is, between tuples of one relation and the tuples of another (others have called these “context-dependent aggregation problems”)—are best treated by representing the sensitive association separately and classifying the individual entities at a low level and the relationship at a high level.
2. Sensitive associations among the various properties of an entity—that is, among the columns of a relation—are best treated by determining those properties that contribute most to the inference and by storing those separately at a higher classification.
3. Properties that appear in a number of sensitive relations may be a better choice for upgrading in general.

DISSECT will use a *minimal upgrading principle* to isolate the minimal set of data which, when upgraded, will remove the identified channels. To the extent possible, we will incorporate expert knowledge and intuition into the channel-removal component of the tool. This is a focus of our current research. Where upgrading is impractical, cover stories can be used [4].

4 Future Work

4.1 Approximate Reasoning for Partial Inference

To address the problem of quantifying the risk of inference of sensitive information, we are using a formal theory of partial inference, called evidential reasoning, which is based on the Shafer-Dempster formalism. The theory defines the notion of partial inference as a probabilistic inference.

Partial inference occurs when it is possible for a user to use low data to infer the truth of high data with some degree of probability. For example, flight destination airports may be sensitive data, while aircraft range, payloads, and departure fields may be stored at a low security level. By combining information about range, payloads, and departure fields, a user may be able to greatly narrow the set of possible destination airports, and in so doing increase the *likelihood* that an aircraft's destination is among the reduced set.

Such probabilistic channels are related to abductive channels because the assumptions and logical rules used in an abductive proof may have degrees of belief associated with them which represent the likelihood that they may be known to or believed by a user. These degrees of belief can then be propagated through the abductive proof tree to determine the degree to which the user is likely to be able to infer the high data in question.

Evidential reasoning explicitly separates background information defining the vocabulary, sorts, and the network of their interrelations from the actual data which describe a particular

situation. Beliefs about actual data values are propagated through the network defined by the relational structures.

4.2 Detecting Data-Related Inferences

Thus far, we have considered only inference channels that can be detected statically, offline, using only the database schema. Detecting additional types of inference channels may require analysis of the actual data stored in the database.

For example, in the above example, the number of flights may be small enough that the association between flight and mission may be guessed or narrowed down. Moreover, if the user can assume a piece of information that is probably correct, such as that the heaviest cargo is probably weapons, and that the mission of a flight carrying weapons is probably war, then some of the sensitive associations will be revealed. In addition, there are well-known methods of statistical attack that can be employed to obtain sensitive information (See [2] for a good overview of these). The sensitive association will also be compromised if the flight and mission records are both sorted in the same order, from which the association can be determined (for example, by a user finding the location of a particular flight and the corresponding mission, and thus making the inference).

In addition, inferences can be drawn from observing the system's changing response to the same query over time. In the example above, whenever a new flight is scheduled, new tuples will be added to both the flight and the mission relations, thus enabling one to infer the mission for the new flight. A simple solution to this problem may be to classify the data for new flights SECRET and then declassify it whenever their numbers had grown sufficiently large. Or, it may be more appropriate to classify the entire mission relation SECRET if the inference threat is deemed to be too high.

4.3 Eliciting Background Knowledge

As we illustrated earlier, it is not sufficient merely to show that none of the high information that is stored in the database can be inferred from the low data that is stored in the database, because much of the information needed to make the inference may be general knowledge about the application that is known to users. In our example in Section 2.1, the fact that a flight's cargo allows a partial inference of the mission is not explicitly stored in the database. This additional information must be elicited from the data designer. The amount of external information that could potentially be brought to bear may be enormous, and exhaustively querying the data designer for possible relationships between every pair of attributes would be time-consuming and burdensome. A part of our ongoing research is directed at developing approaches to enable a focused elicitation of background knowledge that would be of most use in the inference analysis. A learning component that enables the system to incrementally expand its background knowledge will be valuable to the longterm use of this approach.

5 Comparison with Other Research

The LOCK Data Views (LDV) project proposed a *history mechanism* that would use a set of history files and classification rules to detect inference problems during query processing [3]. The mechanism can upgrade both the security level of the process handling the user query and the query result. In addition, the mechanism can block further queries if returning a result would add enough to what is already known to allow an inference to be made. This proposed mechanism restricts itself to what is stored in the database and does not consider inferences that may arise from the addition of external knowledge. This mechanism must also be invoked with each database query; in contrast, we propose a tool that need be used only during data design and not during query execution. It is probably the case that each approach has its advantages in terms of the types of inferences that can be detected.

Thuraisingham has proposed representing data semantics and classifications by multilevel semantic nets [19]. Graph traversal in semantic nets corresponds to limited inference. The classification of implied links found by graph traversal is determined by the classifications of the traversed links. Auxiliary semantic nets can be used to express security constraints. Theoretically stronger representations based on Sowa's conceptual graphs [16] are also suggested.

A method proposed by Hinke was used in TRW's prototype secure database system [6]. His method uses an inference detection tool to detect potential inference problems from the database schema. However, rather than modify the data structures to remove the problem, the method merely runs queries periodically to detect whether there are in fact tuples in the relations in question that can be joined to give a sensitive association. If any data are returned, an inference problem exists. In the TRW prototype, the results of such queries are recorded in an audit trail for subsequent analysis by a security officer, who can then take steps to reclassify some of the data in an attempt to remove the problem.

Morgenstern [10, 11] took the approach of characterizing the inferential closure of a core of unclassified information, with the aim of determining whether any classified information fell within the closure. When this occurs, an inference channel exists. The inferential closure includes all statements for which the relative change from the prior knowledge (expressed as its entropy) for the statement, given statements in the core, is greater than some threshold. The threshold is a parameter that will determine the size of the closure. This work, while theoretically appealing, has proven impractical to implement for realistic problems.

Buczkowski [1] proposed a probabilistic approach, based on Bayesian probability, as a more practical approach to estimating the security risk due to partial inferences. This approach is appealing from a practical perspective, except that a great deal of probabilistic information is necessary that is typically quite hard to estimate precisely [7, 12, 13, 14]. We are investigating evidential reasoning to alleviate these difficulties. Evidential reasoning permits beliefs to be attached to disjunctions of statements, rather than requiring they be assigned to singletons in the universe of discourse and, unlike Buczkowski, we do not require precise probabilities when they are not known.

6 Summary

We have described DISSECT, a new tool being developed for detecting and removing specific types of inference problems in a multilevel database system. We intend for this tool to be used by a data designer to analyze a candidate database schema for potential inference problems. The tool will display a database schema graphically and highlight discovered inference paths, suggesting a minimal set of nodes and links that could be upgraded to remove the problems. Further work is aimed at enabling DISSECT to recognize inference channels leading to partial inferences.

References

- [1] L.J. Buczkowski. Database inference controller. In D.L. Spooner and C. Landwehr, editors, *Database Security III: Status and Prospects*. North-Holland, 1990.
- [2] D.E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Massachusetts, 1982.
- [3] P. Dwyer, E. Onuegbe, P. Stachour, and B. Thuraisingham. Secure distributed data views—implementation specification for a dbms. Interim report a005, Honeywell Systems Research Center and Corporate Systems Development Division, May 1988.
- [4] T.D. Garvey and T.F. Lunt. Cover stories for database security. In *Proceedings the Fifth IFIP WG 11.3 Workshop on Database Security*, November 1991.
- [5] M.L. Ginsberg. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, Los Altos, California, 1987.
- [6] T.H. Hinke. Inference aggregation detection in database management systems. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.
- [7] T.Y. Lin. Commutative security algebra and aggregation. In *Proceedings of the Second RADC Database Security Workshop*, Franconia, New Hampshire, May 1989.
- [8] T.F. Lunt. Aggregation and inference: Facts and fallacies. In *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, May 1989.
- [9] T.F. Lunt, D.E. Denning, Schell, R.R., M. Heckman, and W.R. Shockley. The SeaView Security Model. In *IEEE Transactions on Software Engineering* 16(6), June 1990.
- [10] M. Morgenstern. Security and inference in multilevel database and knowledge-base systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD-87)*, May 1987.

- [11] M. Morgenstern. Controlling logical inference in multilevel database systems. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.
- [12] E.H. Ruspini. Epistemic logic, probability, and the calculus of evidence. In *Proc. Tenth Intern. Joint Conf. on Artificial Intelligence*, Milan, Italy, 1987.
- [13] E.H. Ruspini. Imprecision and uncertainty in the entity-relationship model. In C.V. Negoita and H.E. Prade, editors, *Fuzzy Logic and Knowledge Engineering*, pages 3-17. Verlag TÜV Rheinland, Cologne, 1986.
- [14] E.H. Ruspini. The logical foundations of evidential reasoning. Technical Note 408, Artificial Intelligence Center, SRI International, Menlo Park, California, 1987.
- [15] G.W. Smith. Modeling security-relevant data semantics. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, May 1990.
- [16] J. Sowa. *Conceptual Structures: Information Processing in Minds and Machines*. Reading, Massachusetts, 1984.
- [17] M.B. Thuraisingham. Security checking in relational database management systems augmented with inference engines. *Computers and Security*, 6(6), 1987.
- [18] M.B. Thuraisingham. A nonmonotonic typed multilevel logic for multilevel data/knowledge base management systems. Technical Report MTR10935, The MITRE Corporation, Bedford, Massachusetts, June 1990.
- [19] M.B. Thuraisingham. The use of conceptual structures for handling the inference problem. Technical Report M90-55, The MITRE Corporation, Bedford, Massachusetts, August 1990.

Inference and Cover Stories

Leonard J. Binns

Office of INFOSEC Computer Science

United States Department of Defense
Fort George G. Meade, Maryland

Abstract:

A common use for cover stories is to provide a plausible explanation for an otherwise sensitive event. For example, a plane might be said to carry food when its actual cargo is weapons. Without a cover story, the fact that the cargo is not identified may lead to increased interest from an uncleared user; something which may not be desirable if a mission is to be successful.

Cover stories may also be used to release shades of information. Here, instead of lying they are releasing only sanitized information. At the confidential level a user may be told that a plane is carrying equipment, while a top secret user is told that the plane is carrying electronic equipment.

Cover stories may not always have the ability to protect sensitive information. For example, an uncleared user may have enough world knowledge to discover that a given cover story is not plausible. There is a difference, however, between a cover story that cannot protect sensitive information, and a cover story which itself causes a breach of security. This paper examines cover stories that indirectly disclose the very information they are attempting to protect.

Introduction

Cover stories are plausible explanations which replace gaps of information that the low user would normally see. Gaps that might otherwise cause a curious user to attempt to piece together information for which they are unauthorized. A primary goal of the cover stories is to satisfy the curiosity of an unauthorized user.

In virtually every plausible cover story, however, is some factual information. For example, if Smith is a radar technician and that information is secret, then a plausible cover for Smith would not be "Jones is an engineer." Generally speaking, the object for which a cover story is being developed must be correctly identified¹.

In addition to identifying the object, plausibility often requires other information about the object to be identified. Ensuring that the factual information released is unclassified² is not sufficient, because an attacker can now use this factual information to derive new and possibly classified information on the object via inference. To be sure the cover story does not breach security, it must be shown that all factual information released in the cover story and all inferences possible from that factual information, are unclassified.

This paper focuses on:

- Cover stories
- How an improper cover story can lead to a breach of security, and on
- Recognizing potential inferences caused by cover stories

1. Where the object is not correctly identified, as in someone going "under cover," some attribute(s) of that object must be acknowledged (i.e. height, weight, etc.)

2. or properly classified

Cover Stories That Can't Protect Information

Cover stories are used to protect information. Typically, they give a plausible explanation for information that would not otherwise exist at a user's security level. However, a user may have enough data to piece together what the cover story is trying to protect. When this happens the cover story cannot be relied on for protection, although it may be enough of a deterrent to mislead a portion of the unauthorized users.

In the Mission table (M) shown below, the cover story for flight# C1A2946 is that it is a supply mission, running medical supplies to Europe.

MISSION (M)		(U-TS)	
Flight#	Dest	Cargo	Mission_type
C1A2946	Iraq	Recon Scope 2000	RECON
C1A2946	Europe	Medical Supplies	SUPPLY

Figure 1

This may appear sufficient to convince the novice user that flight# C1A2946 is a supply mission. Figure 2, however, shows that an unclassified user can piece together the fact that flight# C1A2946 is a reconnaissance mission. If this relationship is secret, then the cover story in figure 1 alone cannot protect that relationship. This is an example of a cover story that simply cannot protect information (because the information is available elsewhere). For this particular example, a second cover story³ in either FP, P, or OM is one alternative solution to protecting the secret relationship; a second alternative is to consider redesigning or reclassifying the schema to avoid this type of information flow.

3. The original data would have to be removed or reclassified.

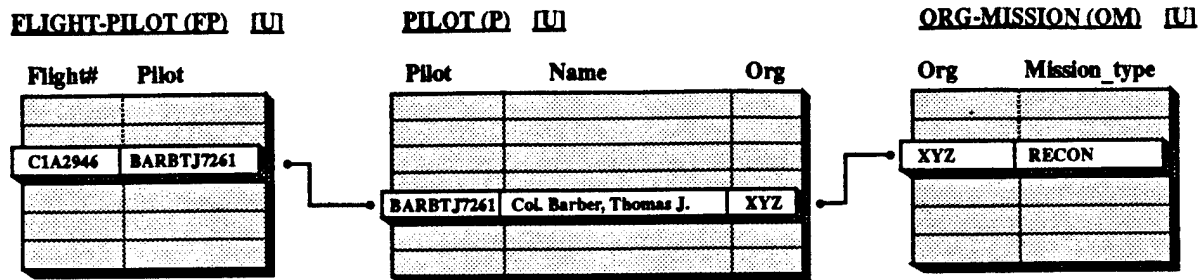


Figure 2

Cover Stories That Breach System Security

In cover stories where some factual data is released, there exists the possibility that a low user could exploit this data to infer high information. The Organization (O) table shown in figure 3 contains the relationship between org "XYZ" and specialty "Russian Language" at the confidential level. At the unclassified level, XYZ's cover story is that its specialty is simply "Language." This cover story is consistent with the user's classification guideline listed in appendix A. In this case, the actual phone number for XYZ was released. Although this is not in direct violation of the classification guideline, it is an indirect breach of security. In conjunction with the database schema shown in figure 4, a low user can use XYZ's phone number to infer its specialty by identifying potential employees of XYZ and their specialty.

This is an example of a cover story which itself causes a breach of security, by releasing information necessary to complete an inference path.

ORGANIZATION (O) [U-C]

Org	Specialty	Phone
ABC	Maintenance	555-1111
XYZ	Russian Language	555-1234
XYZ	Language	555-1234

Figure 3

Plausibility and Usability

Why release factual information in a cover story? It is required in some circumstances to make the cover story plausible. An incorrect attribute may lead a user to question the validity of other attributes in the record, thereby defeating the purpose of the cover story. Unsatisfied with the information, the user may try to retrieve a different answer using an alternative method (i.e. inference). Where the cover story is really a sanitized version of the truth (figure 3) factual information is sometimes required to make the cover story usable.

Although it is always possible to force the user to redesign the database to reduce or possibly eliminate poor schema design, it is our goal to allow these designs as long as their inefficiency does not have an adverse effect on security. Forcing users to adhere to strict design principles can have the affect of driving them away from secure systems altogether. The goal here is to impact the user only when security is at risk, allowing them to work without restriction where possible. The user is responsible for proper classification of data within a record, while the database monitors classification consistency among collections of tables.

As stated earlier, virtually every cover story contains factual information. What must be ensured is that this information cannot directly or indirectly disclose sensitive data to an unauthorized user.

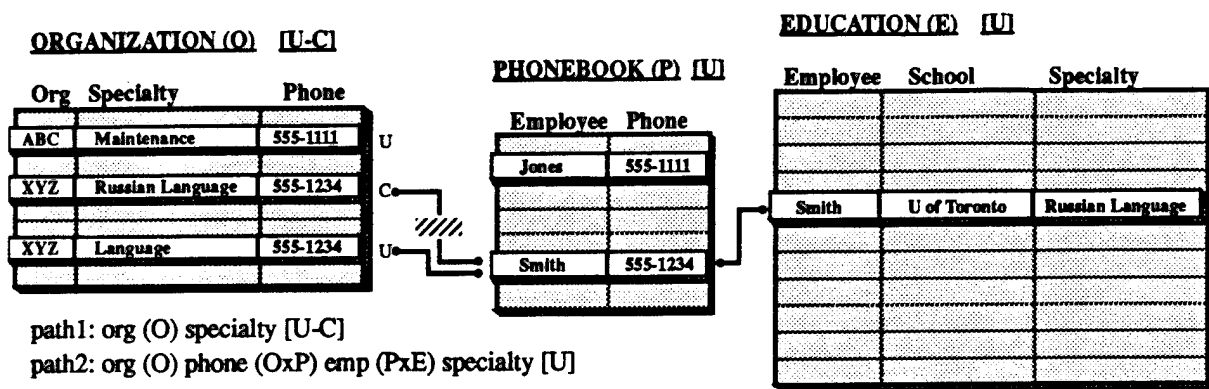


Figure 4

Recognizing Inferences Caused By Cover Stories

The design flaw whereby a cover story opens an inference channel was first introduced in [1]. The channel can be characterized by a relation whose attribute leads to an external attribute which co-exists (and is classified) in the original relation. The definition uses the notion of a path, and level of a path. These are discussed here, and are used in the formal definition that follows.

A *path* identifies the set of attributes and relations used to substantiate a relationship between two attributes; it is a road map showing how the attributes are joined. The smallest path is between two attributes in the same relation, and by definition has a length of one. The Organization (O) table in figure 4 shows the relationship between org and phone. It has a length of one and the path is written *org (O) phone*. More complex relationships use a recursive definition for path. Each join increases the length by one. The relationship between org and employee in figure 4 is substantiated by joining the Organization and Phonebook relations. The path has a length of two and is written *org (O) phone (OxP) employee*. Cyclical paths are not allowed; neither tables nor attributes can be revisited in a path.

A path of length ($n=1$) is defined as:

$$P(a_0, a_1, A, R) = [a_0 (r_1) a_1 \mid \\ \wedge a_0, a_1 \in A \\ \wedge a_0 \neq a_1 \\ \wedge r_1 \in R \\ \wedge a_0, a_1 \in r_1 \\ \wedge \text{Cardinality}(A) = 2 \\ \wedge \text{Cardinality}(R) = 1]$$

A path of length ($n > 1$) is defines as:

$$P(a_0, a_n, A, R) = [P(a_0, a_{n-1}, A-a_n, R-r_n) (r_{n-1} \times r_n) a_n \mid \\ a_0, a_{n-1}, a_n \in A \\ \wedge a_0 \neq a_{n-1} \neq a_n \\ \wedge r_{n-1}, r_n \in R \\ \wedge r_{n-1} \neq r_n \\ \wedge a_{n-1}, a_n \in r_n \\ \wedge a_{n-1} \in r_{n-1} \\ \wedge P(a_0, a_{n-1}, A-a_n, R-r_n)]$$

Where

A is a set of $n+1$ attributes, and

R is a set of n relations

The *level* of a path is defined by the relations used in traversal; it is composed of the path's hierarchical and non-hierarchical security levels unioned together. The hierarchical level is the least upper bound of all the hierarchical levels encountered in the path. The non-hierarchical level is the union of all non-hierarchical levels encountered.

The level of a path p at time t is defined as:

$$L(p,t) = [L_h(R,t) \cup L_c(R,t) \mid \exists a_0, a_n, A [p = P(a_0, a_n, A, R) \wedge t \in \text{time}]]$$

Where:

$$L_h(\text{nil}, t) = U$$

$$L_h(R, t) = [\text{Level}(r, t) \mid r \in R \wedge \text{Level}(r, t) \geq L_h(R-r, t)]$$

$$L_c(\text{nil}, t) = \text{nil}$$

$$L_c(R, t) = [\text{Comp}(r, t) \cup L_c(R-r, t) \mid r \in R]$$

$\text{Level}(r, t)$ = Hierarchical security level associated with relation r at time t

$\text{Comp}(r, t)$ = Non-hierarchical compartment(s) associated with relation r at time t

R = set of relations

$\{U, C, S, TS\}$ = Hierarchical security levels, and $U < C < S < TS$

We say that a cover story is the cause of inference if it releases information that could be used to yield the true state of what the cover story is designed to protect. Formally stated, potential inference through the use of a cover story is defined¹:

$$I_c(a_i, a_j) = [\text{TRUE} \mid \exists p_1, p_2, a_k, r, A_1, A_2, R_1, R_2, t$$

$$[p_1 = a_i(r) \wedge a_j = P(a_i, a_j, A_1, R_1)$$

$$\wedge p_2 = P(a_i, a_j, A_2, R_2)$$

$$\wedge a_i \neq a_j \neq a_k$$

$$\wedge a_i(r) \wedge a_k \in p_2$$

$$\wedge \neg(L(p_1, t) \leq L(p_2, t))]]$$

Looking back at our example in figure 4, we see that *specialty* (a_j) is the attribute that is both external and local to Organization. *Phone* (a_k) is the attribute or "hook" that can be used in a path leading to *specialty* outside of Organization. Using our definition, we see that a potential inference does exist. The values used to substantiate this are shown below.

1. In this context, \in is used to denote a subpath.
i.e. $\text{org (O) phone} \in \text{org (O) phone (OxE) emp (ExP) specialty}$

$a_i = \text{org}, a_j = \text{specialty}, a_k = \text{phone}$

$A_1 = \{\text{org, specialty}\}$

$A_2 = \{\text{org, phone, emp, specialty}\}$

$r = O$

$R_1 = \{O\}$

$R_2 = \{O, E, P\}$

$p_1 = \text{org (O) specialty}$

$p_2 = \text{org (O) phone (OxE) emp (ExP) specialty}$

$\text{org (O) phone} \in p_2$

$\neg(L(p_1=C) \leq L(p_2=U))$

$\therefore I_c(\text{org, specialty}) = \text{TRUE}$

Notice that we assign path classifications to suit our needs. The only constraint is that the levels assigned are consistent with the range of possible values. For example, Organization can be assigned either unclassified (U) or confidential (C) security levels however Phonebook is strictly unclassified. The fact that there exists a potentially classified path p_1 and a potentially unclassified path p_2 is a necessary ingredient to show the design is inherently flawed and could potentially breed inference.

The actual tuple values shown in figure 4 are not used when determining the soundness of the design. They are used here to illustrate how a poor design could lead to an inference path, via specific database instance.

A database is said to be free of potential inference through the abuse of a cover story if for all attributes a_i and a_j , $I_c(a_i, a_j)$ is false.

Where it is infeasible to eliminate potential inference paths, run-time analysis would monitor the contents of the database based on design-time analysis. Run-time analysis would substantiate when a potential inference path becomes an actual inference path.

Relationship to Other Classes of Inference

A database where $I_c(a_i, a_j)$ is false for all a_i and a_j is by no means inference free; it only means that a cover story cannot be used against *itself*. The database is still subject to other classes of inference; there is no guarantee that the information released in a cover story could not be used to exploit some *other* type of inference channel, hence the need for additional analysis. The inference definition presented here identifies a specific class of inference. It depends on a set of complimentary tools which detect other classes of inference, in order to form a comprehensive inference policy. Such a policy is required if we are to trust multilevel databases to truly protect the information it manages.

Summary

Cover stories, whose intention is to protect information, can fail in two ways. They can take either an active or passive role in the disclosure of information to the unauthorized user.

Garvey[4] recognized that a cover story cannot protect information if the user has enough data to piece together via inference the information it is trying to protect. In this case, the cover story is taking a passive role in the disclosure of information to the unauthorized user. To counter this he indicates the need to identify such situations and provide additional cover stories to block the inference path(s) which jeopardize the initial cover story.

Plausibility of a cover story may require some factual, non-critical information to be released. Where factual information is released, there is the possibility a hostile user could abuse the information to obtain critical (high) data by using it to complete an inference path. Here,

the cover story is taking an active role in the disclosure of information to an unauthorized user. This paper focuses on detecting whether a cover story can be used to disclose information itself is trying to protect. This paper does not address the impact a cover story has on other information in the database which the cover story does not directly protect; i.e. this paper does not address the impact a cover story has on other classes of inference. Such considerations are being addressed separately, with the long term goal of developing a policy that *does* address a range of inference classes.

Finally, the definition of $I_c(a_i, a_j)$ would presumably check the entire database to determine if a cover story could exist and could potentially be used to divulge the relationship between a_i and a_j . Alternatively, one could modify the definition to use it as a specific tool in developing cover stories. Passing r as an argument, rather than testing to see if there exists some r that would satisfy the equation, would provide a useful tool to someone creating a cover story in a specific relation. Additionally, instead of returning TRUE the definition would return the offending piece of information:

$$\begin{aligned} I_c(a_i, a_j, r) = \{a_k \mid \exists p_1, p_2, A_1, A_2, R_1, R_2, t \\ [p_1 = a_i(r) \wedge a_j = P(a_i, a_j, A_1, R_1) \\ \wedge p_2 = P(a_i, a_j, A_2, R_2) \\ \wedge a_i \neq a_j \neq a_k \\ \wedge a_i(r) a_k \in p_2 \\ \wedge \neg(L(p_1, t) \leq L(p_2, t))]\} \end{aligned}$$

Applying this to the cover story we wish to pose for specialty in the Organization relation of figure 4:

$$I_c(\text{org}, \text{specialty}, \text{O}) = \text{phone}$$

This indicates that if the actual phone number is supplied in a cover story for Organization, it is possible a hostile user could complete an inference path between *org* and *specialty* using *phone*; to prevent this, a cover story for phone number must be provided.

References

- [1] Binns L.J. "Inference Through Polyinstantiation," *Proceedings of the Fourth RADC Database Security Workshop*, April 1991.
- [2] Binns L.J. "Inference Through Secondary Path Analysis," *Proceedings of the Sixth IFIP WG 11.3 Working Conference on Database Security*, August 1992. (Submitted)
- [3] Garvey T.D., Lunt T.F., and Stickel M.E. "Abductive and Approximate Reasoning Models for Characterizing Inference Channels," *Proceedings of the Fourth Workshop on the Foundations of Computer Security*, Franconia, NH, June 1991.
- [4] Garvey T.D., Lunt T.F. "Cover Stories for Database Security," *Proceedings of the Fifth IFIP WG 11.3 Working Conference on Database Security*, November 1991.
- [5] Hinke T.H. "Inference Aggregation Detection in Database Management Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.
- [6] Lunt T.F. "Aggregation and Inference: Facts and Fallacies," *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1989.
- [7] Lunt T.F. "The True Meaning of Polyinstantiation: Proposal for an Operational Semantics for a Multilevel Relational Database System," *Proceedings of the Third RADC Database Security Workshop*, June 1990.
- [8] Lunt T.F. "Polyinstantiation: an Inevitable Part of a Multilevel World," *Proceedings of the Fourth Workshop on the Foundations of Computer Security*, Franconia, New Hampshire, June 1991.
- [9] Morgenstern M. "Controlling Logical Inference in Multilevel Database Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.
- [10] Thuraisingham B. "The Use of Conceptual Structures for Handling the Inference Problem," *Proceedings of the Fifth IFIP WG 11.3 Working Conference on Database Security*, November 1991.

Appendix A

Mock Classification Guideline

- An organization's name alone is not classified
- An organization's specialty is not classified unless stated otherwise
- The *relationship* between organization and specialty is confidential if its specialty is one of the following:
 - Russian Language
 - Metalinguistics
 - Optical Fiber Transmission
 - Civil Engineering

If the specialty falls within these classified areas, the following unclassified specialties are to be used when referencing that organization to an unclassified user:

<u>Classified Specialty</u>	<u>Unclassified Specialty Description</u>
Russian Language Metalinguistics	Language
Optical Fiber Transmission Civil Engineering	Engineering

- To provide consistency for the unclassified user, any record which relates organization and specialty at the confidential level must be polyinstantiated at the unclassified level. The record at the unclassified level will reflect the organization's unclassified specialty.

AERIE: An Inference Modeling and Detection Approach For Databases

Thomas H. Hinke and Harry S. Delugach

Department of Computer Science
Computer Science Building
University of Alabama in Huntsville
Huntsville, AL 35899

Abstract

This paper describes initial research to develop a new inference model and detection approach for addressing the database inference problem. The model provides a means of classifying various types of inferences, with each class having various inference detection methods. The paper then provides examples of each type of inference class.

To address the problem of the extensive knowledge that may be brought to bear on an inference objective by an adversary, the model recognizes various classes of information and shows how this information can be integrated into an inference system. The nature of the information required for a particular inference can be used to provide guidance to the database designer as to what data should be protected to prevent the inference.

1. INTRODUCTION

With the publication of the Trusted Database Interpretation of DoD 5200.28-STD, a big step has been taken to provide protection for large amounts of data stored in databases. While the ability to provide discretionary and mandatory access control (at least at the higher evaluation levels) is a major step forward in providing protection for the database, it is not sufficient. An organization could properly use the protection facilities provided by a multilevel secure database management system and still be at risk, due to the ability of an adversary to access unclassified data and be able to deduce more highly classified data using inference techniques.

This problem will become more acute as more data is placed under the control of databases and these databases are connected to networks, such that they can be accessed by large groups of users located all over the world. In addition, as network accessibility becomes the norm for database access, a particular user could potentially have access to a large number of databases. If the user was working on behalf of an adversary, this adversary could then have access to considerably more data than has been available to a single user in the past. This greatly multiplies the potential to infer unauthorized data, since there is a greater pool of available data with which to perform the inference.

Previous inference research has been characterized by [Hinke90b] into the following categories:

1. "Efforts to discover fundamental laws that determine whether the potential for undesirable inferences exists within a given database;
2. Efforts to discover automatically inference rules from fundamental relationships among data that pertain to a domain, and
3. Efforts to automate (via expert systems) the process of inferring sensitive data within a specific domain."

This will be extended with a fourth category to reflect some recent work at SRI International.

4. Efforts to jam the inference channel with "noise" provided by plausible cover stories.

Research into the discovery of fundamental laws that determine whether the potential for undesirable inferences exists include work on statistical databases [Denning82, Cox88, Matloff88] and work exploring the inference issues with respect to functional and multivalued dependencies within relational databases [Su90]. Also included within the first category is the work of Morgenstern [Morgenstern87, Morgenstern88] to propose a theoretical foundation for inference, using such concepts as a sphere of influence that is the transitive closure of all that can be inferred by a particular fact.

Research in the second category is represented by the work of Hinke [Hinke88, Hinke90a], which sought to discover inference channels that would permit classified relationships between two entities to be discovered by finding second paths, consisting of relationships between other entities that could be used to make the sensitive linkage. One of the important results of this work was that the specific rules required to find the classified relationship did not have to be stated explicitly. While they could be stated upon viewing the second path discovered, all that was required to perform the inference discovery was to find this second path by traversing relationships between entities until the two target entities were joined, using a path that was less classified than that associated with the direct relationship between the two entities. Also included

in this area is the work of Thuraisingham [Thuraisingham91], which uses conceptual structures to represent multilevel applications. The conceptual structures can be used to describe a database and this description used to determine if an adversary has the ability to draw unauthorized inferences. The use of abductive and approximate reasoning for discovering inference channels has been investigated by [Garvey91b].

The third category of inference research seeks to capture rules that could be used by an expert to detect inference problems within existing databases. Research in this area includes the work by Ford Aerospace on their inference controller [Buczowski90].

The fourth category of inference work is represented by the current research on inference underway at SRI International [Garvey91]. While the work addresses the general inference problem, one of the areas that they considered in their published paper was the use of cover stories to add noise to the inference channel. This is especially useful when the inference channel may be composed of widely known facts that cannot be easily classified.

The research described in this paper builds on the work in the second and third categories, by proposing the use of a new inference model developed at the University of Alabama in Huntsville. This model, called the AERIE (Activities, Entities and Relationships' Inference Effects) Model, seeks to permit the discovery of inferences using inference targets (sensitive things that are to be protected from unauthorized disclosure through inference). It also proposes utilizing a knowledge-base, using conceptual graphs, that will permit an Inference Analysis Tool to reason about the existence of information not only within the database, but also other knowledge that would be assumed to be known by an adversary.

Conceptual graphs are based on first-order logic, as denoted by Charles Peirce's existential graphs from the late 1800's. An extension of semantic networks, they provide a powerful, extensible means of capturing real-world knowledge, such as the difference between class types and instances of a class; multiple constraints on the same individual or class, and inheritance of type characteristics from a supertype. The advantage of using conceptual graphs in this research is that they allow modeling of information without being codified into rules; i.e., knowledge can be applied in flexible ways as needed. Conceptual graphs are being considered as a standard for knowledge interchange by the ANSI X3H4.6 task group on conceptual schema's IRDS committee (Information Resource Dictionary Systems).

This current work builds on the work of [Hinke88, Hinke90a, Thuraisingham91]. The work of [Hinke88, Hinke90a] is believed to be the first proposal to use conceptual structures for addressing the inference problems. This initial work was extended considerably by [Thuraisingham91], who proposed a multilevel semantic net and associated rules of deduction that could then be used to prove whether or not a multilevel semantic net satisfies desired security constraints. In the same paper, she extends these ideas to multilevel conceptual graphs and discusses a reasoning strategy that can be used to determine if the security constraints are violated in a particular conceptual graph. The AERIE research is based on conceptual graph structures and will be able to use the inference techniques suggested by [Thuraisingham91]. However, it extends the work of [Thuraisingham91] by proposing a classification of inference targets and a two-phase inference approach. Each of these differences will be briefly discussed here so that the contribution of AERIE, vis-a-vis related work, can be put into perspective.

An inference target class is a type of real-world inference based on the nature of the inference that an adversary may attempt to perform. These classes can be related to real-world entities, activities and relationships. AERIE has identified a number of such classes. The value of the class is that it provides a focus for identifying appropriate inference techniques. For example, one of the contributions of the AERIE research is its use of the part-of relationship as a means for performing inferences of the Entity target class. Since real world entities have a physical realization, and many physical things have parts, the identification of a unique part for a particular entity could provide a strong suggestion that the data concerns the entity of which

the assembly is a part. This type of inference would not, however, be appropriate for the Activity inference class.

The two-phase inference approach, consisting of the materialization of things and then the determination of relationships between these materialized things, suggests that different types of methods may be appropriate to each phase. For example, statistical techniques may be used to assist in the materialization, and thus be combined with the purely logical approaches proposed by [Thuraisingham91] for reasoning about relationships among materialized things.

The remainder of this paper is organized as follows: Section 2 describes the AERIE Model, while Section 3 describes our proposed architecture for an Inference Analysis Tool (IAT) that can implement the model. Section 4 provides the current status of the project and the conclusions to date.

2. AERIE MODEL

This section will describe the features of the model and provide some examples of each inference target class.

2.1. INFERENCE TARGET CLASSES

This research will investigate the database inference problem in the context of the AERIE Model, a new inference model that is under development at the University of Alabama in Huntsville.

The AERIE Model views inference detection as a two-phase process. During the first phase, "things" of interest are materialized from the database. By materializing something, we mean "detecting" it, much as sonar is used to detect the presence of a submarine. In this research, we are using various techniques to detect the presence of various "things" in the database, such as a particular type of aircraft or a construction project at a sensitive location. The reason the word "materialization" is used is that, in effect, all inference is making visible something that is not clearly articulated in the database. If it had been - and it represented sensitive data - then presumably it would have been properly protected. However, since it is not clearly articulated, it may not be properly protected, and hence is available for an adversary to materialize the information and thus gain access to classified information.

Those "things" that have a realization as a physical object are called entities. These are analogous to nouns in the English language. Those "things" that describe some action or state of being in the sense of English verbs are called "activities".

The second phase of inference detection is the determination of relationships between various materialized entities and/or activities. These relationships can be used to perform intermediate inferences that are required to determine if a desired inference can be made. Or, the relationships themselves may be sensitive and hence, the target for the inference.

One of the fundamental beliefs underlying this research is that the prevention of undesirable inferences is a counterpart to the problem of providing a means for someone to perform them. Thus, in order to prevent undesirable inferences, we must be able to determine if they can be made. The problem is that failure to determine if an inference can be made could be an indication that the inference cannot in fact be made, that more time is required for the tool to make the inference or that the tool is inadequate to make the inference, but a better tool could. At this time, inference detection should be viewed as an imperfect method to assist a system security officer in increasing the security of the data under his or her control. If an undesirable inference is detected, then it can be removed. However, it may not be possible to discover all undesirable inferences.

Since what constitutes an undesirable inference is highly dependent upon the nature of the data in a particular database and an assessment of the information available to one's adversaries, this research assumes that one begins with a set of inference targets that the system security office has determined to be undesirable. These targets may fall into one or more of the inference target classes that will be presented shortly. These targets represent potential undesirable inferences that we want to ensure cannot be obtained in the database under study using data of a lower classification than the target inferences. If such targets can be inferred using data of a lower classification, then the database contains an inference vulnerability that should be eliminated by classifying some data item in the inference chain so as to break the chain at the lower level of classification, or adding "noise" in the form of cover stories, as suggested in [Garvey91].

In the AERIE Model, the inferences are presented in terms of inference target classes, materialization method classes and materialization method instances. An inference target class represents the inference objective of an adversary, and thus the protection objective of a system security office (SSO). A materialization method class represents the class of methods that can be used to perform particular types of inferences. A materialization method instance represents a particular member of a materialization method class.

To date, the research on the AERIE Model has identified seven inference target classes. In addition to assigning each target class a number for reference purposes, each class can be characterized in terms of the entities, indicated with an "E," and/or activities, indicated with an "A," that are involved in the inference. In one case, the inference target class is specified in terms of previously identified target classes, and these are indicated with a "C". Where unknown entities or activities are involved in the inference target class, these are indicated with the letters W, X, Y and Z. Using this notation, the seven inference target classes are as follows:

- Class 1, E: The materialization of an entity;
- Class 2, A: The materialization of an activity;
- Class 3, (E,E): The materialization of a sensitive relationship between two or more materialized entities;
- Class 4, (A,A): The materialization of a sensitive relationship between two or more materialized activities;
- Class 5, (E,A) or (A,E) (we will assume that they are equivalent): The materialization of a sensitive relationship between one or more materialized entities and one or more materialized activities;
- Class 6, ((W,X),(Y,Z)): The materialization of a sensitive relationship between sensitive relationships, and
- Class 7, [C1,C2,...,Cj] => C: The materialization of a sensitive rule from existing classes.

The Class 1 and 2 inferences are the simplest, since they deal with the inference of only a single entity or activity. The Class 3, 4 and 5 inferences deal with relationships between entities and activities, and thus are of a more complex nature than the first and second classes. (However, Classes 3, 4 and 5 are believed to be similar in complexity.) The Class 6 inference deals with relationships between relationships and is more complex than the other five classes, since it includes these more primitive classes as well as the additional requirement that the earlier relationships be related to each other. Finally, the Class 7 inference is somewhat distinct from the others, since it deals with the ability to infer rules from previously known classes or class instances. More will be said about the Class 7 inferences later.

These inference target classes play a dual role. While they have been identified in the context of classes of inference targets, they can also represent data that has been materialized, either at some initial state (since it represents data stored in the database), or because it has been materialized in some previous inference processing stage. Thus we can talk of Class 1 data as data about entities, Class 2 data as data about activities, Class 3 data as data about relationships between entities, etc. These concepts of data classes and inference target classes merge, since from an inference processing perspective it makes little difference whether the specified class of data was originally stored in the database or has been materialized through inference processing.

2.2. EXAMPLES FOR EACH TARGET CLASS

Having provided an overview of the AERIE Model, its various components, the inference target classes and their related data classes, we will now describe an example of each inference target class, along with the methods that are used in performing the inference. Following the presentation of these examples, we will show how these methods can be described in terms of materialization method classes.

An example of a Class 1 inference target can be found within a logistics database. If a site orders a part that is unique to a particular type of equipment, such as a certain radar unit, then an adversary with access to this database could infer that the site has this particular type of radar unit. This inference is made using the following method: $E1 \text{ AND } (E1, E2) \Rightarrow \{E2\}$, where $E1$ is some unique part and the $(E1, E2)$ relationship is the part-of relationship that breaks down the parts that are contained in each piece of equipment. The set $\{E2\}$ represents all of those end-products in which the part is used. If the part is used only for a single end-product, then the cardinality of the set $\{E2\}$ is one, and we have an inference that results in the unique identification of a piece of equipment.

While the previous entity materialization example has been described in terms of using a logical inference method, the entity materialization could also be accomplished using statistical inference methods, if statistics were provided by the database. The statistics could be probed to isolate a particular, identifiable member of the group covered by them. The end result is the materialization of a particular entity; hence, statistical inference methods are included under the first inference target class, since they can be used to materialize a particular entity.

An example of a Class 2 inference target is the ability to infer that a construction project is occurring based on the class of equipment that is being ordered. Thus, if an equipment requisition includes equipment that can be used collectively for digging, pushing (e.g., pushing dirt) and carrying, this could indicate the existence of a construction project. On the other hand, if the equipment being ordered is used for mowing or harvesting (e.g., wheat), then this would not be an indication of a construction project. To perform this inference requires that a construction project be modeled with a definition that characterizes it as including the activities of digging AND pushing AND carrying. Then, the various equipment parts must be characterized with Class 5 data that has the form (E, A) , relating the parts to the activities that

they support. For example, a blade supports the activities of pushing, a bucket supports the activities of digging and a loader supports the activities of carrying and pushing. Finally, pieces of equipment must be characterized in terms of their component parts, with Class 3 (E,E) relationships that associate a part with an end-item. Thus, a backhoe contains a bucket and a loader, while a tractor contains a blade.

The inference of a construction project activity can now be made by determining whether there exists a requisition that contains equipment that can be used collectively for digging AND pushing AND carrying. To perform this inference analysis, the parts in an order can be used along with the (E,E) and (E,A) relationships to perform an inference using the following methods:

- Definition, to determine the definition of a construction project, which would be listed as an instance of the inference target Class 2;
- Method A1, (A1, E1) => E1 to determine the equipment components that can be used for digging, pushing and carrying;
- Method E1 AND (E1,E2) => E2 to infer pieces of equipment and component parts that are associated with the components that can be used for digging AND pushing AND carrying.
- These various E2's will be checked against the order and used with the previous rule to generate more component parts and end items that are related to the entities that perform the digging, pushing and carrying activities associated with a construction project. For this inference, we have used two methods and a definition.

An example of a Class 3 inference target is the determination of a sensitive association between two entities. For example, assume that some organization was attempting to keep its area of operation secret. This would be an association between the entities of organization and location. Now assume that this organization ordered blades. If these blades were for bulldozers, then one could not make much of an inference. However, if these were snow blades, then one could make the inference that the organization operates in snow country. While this is admittedly not a precise location, it does narrow down the possible area of operation.

A Class 4 inference represents a sensitive association between two or more activities. For example, a new type of tank could be under development. Characteristics of this tank could be revealed if the designer of the tank ordered both blades and a bucket, indicating that this tank could participate in the associated activities of pushing and digging. The fact that these activities could be combined might be sensitive information. This inference was made by using the method: E1, (E1,A1) => A1.

A Class 5 inference represents a sensitive association between one or more entities and one or more activities. For example, if an intelligence activity required that a certain fixture be placed in the space shuttle's payload bay to support a particular type of sensor, then the association of this fixture with the intelligence gathering activity would represent a sensitive relationship that should be protected.

A Class 6 inference represents a sensitive relationship between sensitive relationships. An example of this class is a student grade inference. Assume that grades are posted by student numbers, to preserve the confidentiality of the grade that a particular student received. This represents a Class 3 relationship between the entity student and the entity grade (e.g., (E,E)). However, if these posted grades are sorted by the last name of the student, then this represents a sensitive relationship called "Sorted-by-name" between the (Student_number, Grade) relationship, which is public knowledge and the (Student_name, Grade) relationship, which is

sensitive. However, if this Sorted_grade relationship can be inferred, then the very sensitive (Student_name, Grade) relationship can also be inferred.

A Class 7 inference represents the inference of a sensitive rule. An example of a sensitive rule might be one used by a credit card company that approval to all charges for food, using the reasoning that since the food is already consumed, there is no reason not to approve it. In general, this class of inference incorporates any inference that results in a rule, rather than an entity, attribute or relationship. This class of inference target represents a considerably different target than the previous ones that are shown, but it is included since it represents another type of information about which one could launch an inference attack.

2.3. INFERENCE METHOD CLASSES

For each inference target class, one or more methods will apply to performing the inference. These various methods can be clustered into classes, such that each member of the class has the same basic characteristics. It is also the case that method classes applicable to one inference target class may also be applicable to others.

Consider, for example, Class 1, the entity target class. The method classes that are applicable to inferring an entity include the following:

1. Statistical inference;
2. E, (E,E), and
3. A, (A,E).

The inference method classes applicable to inferring an activity include the following:

1. Statistical;
2. E, (E,A), and
3. Traffic flow analysis (analogous to the traffic flow analysis performed for networks, but for example concentrating here on a high volume of activity on data about Iraq).

The inference target class (E,E) has the following method classes that are applicable:

1. Second path analysis, as proposed in [Hinke88];
2. Those methods required to materialize an E that may exist on the path; hence, all of the methods applicable to Class 1 inference targets, and
3. Perhaps some paths will utilize (E,A) target classes as a means of linking entities, but this currently remains an open issue for the research.

These examples illustrate one of the areas of investigation being pursued by this research, with the goal of identifying a more complete set of method classes, and then showing target classes to which they are applicable.

3. INFERENCE ANALYSIS TOOL

While the primary focus of this research to date has been on the model, some initial thought has also focused on the nature of the tool that can be used to detect inferences. This tool, called an inference analysis tool (IAT), will be used to provide guidance to a database designer or administrator with respect to inferring sensitive information from the database. Given a semantic description of the database, along with some amount of information specific to the database's domain, the IAT should be able to determine whether certain items of classified information could be inferred from the database. Within the paper, we will develop a preliminary architecture, consisting of tool components, and a proposed sequence of steps to provide assistance to database designers and administrators.

The next section will describe an inference analysis tool, and the following section will describe how that tool could be used to materialize an example inference - in this case a sensitive activity.

3.1 INFERENCE ANALYSIS TOOL ARCHITECTURE

One value of the tool consideration is that the clarity of our model can be enhanced if we question how it can be used in a practical tool. This will provide a solid algorithmic basis for methods we develop based on the model. Furthermore, by considering an automated tool along with the model, we can better identify what parts of our inference analysis can be automated, what the difficulties are and what fundamental limitations exist regarding automated inference in general.

An adversary might possess a large body of generally available knowledge that he may bring to bear in attempting to draw inferences from a given database. In order to predict these inferences, a tool must encode an amount of knowledge that is roughly equivalent to the adversary's. The tool makes use of information obtained from several sources. We presume that this information is in the form of conceptual graphs. The knowledge sources are:

- Description of the database under analysis. There are two parts to this:
 - Database specification graphs (*DBSG*): describe the classes of information that are immediately available (i.e., directly inferable), and
 - Database instance graphs (*DBIG*): capture actual instances in the database (if known) that are facts in the inference analysis.
- Domain-independent knowledge (*DIK*): General knowledge (i.e., information that is considered publicly available), either so-called "common sense" knowledge, or encyclopedic knowledge. (We discuss this further below.)
- Domain specific knowledge (*DSK*): Information that a domain expert would know.

The above will be called *knowledge sources*. In addition, the tool should have access to the following:

- Sensitive targets (*ST*), which are instances of some inference target class. The database designer supplies facts and/or relationships that he wants to be protected against being inferred from the database.

- The *AERIE Model*, described above, that guides the IAT's steps in seeking and identifying inferences between the source information and the database designer's sensitive targets.

The overall goal of the inference analysis process is to determine whether or not sensitive targets can be inferred from the original database, and if so, what information enables the inference to be performed. Since the analysis tool will be based upon the conceptual graph representation, each sensitive target will be kept as a graph. The sequence of inference steps that derives a sensitive target graph from the database and other graphs will be called an inference path. An inference path therefore contains the originating information as well as any intermediate inferences (and their enabling graphs).

The tool maintains a set of inference paths that it has currently derived. A *stored inference path* is a sequence of one or more inference steps. An *inference step* is composed of the following:

- *Enabling graphs*. A set of graphs (identified as to their origin) that enables the inference to be performed;
- *Inferred graphs*. A set of graphs representing the facts that were derived;
- A list of the materialization methods used to perform the inference;
- A characterization of the materialization method class(es), and
- A characterization of the target class(es).

Within an inference path, an inferred graph for one step may also serve as an enabling graph for a subsequent step. We define a *path-enabling graph* to be an enabling graph that is not also an inferred graph of some other step in the path. The inference path can therefore be envisioned as a function whose input consists of path-enabling graphs, and whose output is the inferred graphs of the last step in the sequence.

The tool's basic operation is to construct inference paths. Once all the source information is accessible (see above), the tool starts out with an empty set of current inference paths, indicating that no inferences have yet been derived. For each sensitive target, the tool seeks an inference path whose path-enabling graphs appear in one of the knowledge sources. If no such path-enabling graphs are found, then the tool postulates some intermediate graph(s), and seeks an inference path that can infer the intermediate graph(s), which can then be connected to a previous path.

Once an inference path is established, different interpretations can be drawn and different advice to the database designer/administrator will result. For example, if all path-enabling graphs for a piece of sensitive information appear in the general knowledge base, then the sensitive information is based entirely upon common knowledge. It will then be impossible to maintain the information's security, except through the possible use of cover stories. Likewise, if all path-enabling graphs are in either the general or domain-specific knowledge base, then the information can be inferred by a knowledgeable specialist, and will be difficult to keep secure. If at least one path-enabling graph appears in the database specification, it would show that mere knowledge of the database's structure allows the sensitive information to be inferred.

The AERIE Model gives a taxonomy of target classes, where each class consists of one or more method classes, and each method class consists of one or more methods that lead to inference of the target. The tool uses the AERIE Model to organize its search and storage of information. The tool starts by classifying each sensitive target into one or more target classes. For each of these, the model has one or more materialization method classes that are used to infer it. Each method class has one or more actual methods that can be applied one by one. The

order of search corresponds to the order in the model: The tool seeks graphs for which the materialization method will produce either the sensitive target or some intermediate target.

The following discussion describes examples, to give the "flavor" of the IAT. These examples make use of a sample database that consists of an equipment manufacturer's inventory and shipping information. The hypothetical database contains records with information such as part number, description, quantity on hand and destination.

The domain-independent knowledge base (DIK) of general knowledge consists of the following:

- There is type hierarchy representing a set of classes, where each type is a subtype (sub-class) to the class of its parent(s). The type T at the top represents the universal type. The existence of a type hierarchy implies the inference rule that if some instance X is of type A and type A is subtype to type B, then instance X is of type B also.
- If some instance Y is a part of some instance X, and some instance Z is a part of the instance Y, then instance Z is a part of instance X.
- A piece of equipment exists with the following typical associated information:
 - Cost (in money);
 - Operated by a person;
 - Located in some place;
 - Owned by either an organization or a person, and
 - Part of something else.
- "Pushing" (which is an act) is typically caused by some animate instance Y and acts upon some instance X.
- If some instance X exists, and some Y is a part of X, then an instance of Y exists.

The domain-specific knowledge base (DSK) consists of the following:

- A construction project (an activity) exists with the following typical associated information called a *schema*:
 - Cost in some financing;
 - Located in some place;
 - Employs one or more persons, and
 - Has as its parts digging, carrying or pushing.
- A blade is used for pushing, a bucket is used for digging, a hitch is used for pulling and a loader is used for carrying or pushing.
- A tractor has as its parts a hitch, a blade and a tire.
- A backhoe has as its parts a loader, a bucket and a tire. A bucket has as its part a tooth.
- A cotton harvester has as its parts a hitch and a tire.

With this information as background, the next section will consider how the IAT could use this information to discover an inference problem in a database.

3.2 MATERIALIZATION OF A SENSITIVE ACTIVITY

Suppose the database designer decides that a construction project is a sensitive activity -- that is, the existence of any construction project is deemed sensitive. Assume that a heavy equipment manufacturer's database is to be analyzed. This database would include both end items (such as tractors) and parts shipped to dealers and customers. Assume further that the knowledge-bases describe previously are available to the inference analysis tool. The tool proceeds as follows to see if it can materialize a construction project.

To perform the analysis, the tool must identify what (if any) knowledge might be used to infer the existence and/or location of a construction project. First, the *DIK* is scanned for any appearance of *CONSTR_PROJECT*; there is none. Then the *DSK* is scanned. Since it contains a schema for *CONSTR_PROJECT*, the existence of a construction project can be inferred if some of its associated concepts are present (e.g., *FINANCING*, *PLACE*, *PERSON*, *DIGGING*, *CARRYING* or *PUSHING*). Assume that one of these concepts, *PLACE*, is found in the specification graph for the database. If we were to decide that the existence of one concept in the specification graph is sufficient to infer the rest of the schema, then we would have just shown that the location of a construction project could be inferred from the database.

The reader can quickly see that such an inference is not particularly useful. As an old saying goes, "Everybody's got to be somewhere," so that mere sharing the concept of location (as opposed to sharing some particular location instance) is insufficient to infer that it is a construction project that exists at the place. To address this problem, some heuristic must be chosen that determines what components of the schema must be present before we can infer the central concept (in this case, *CONSTR_PROJECT*).

If we adopt some general heuristic that at least half of a schema's components must exist before we are willing to infer the entire schema, we immediately encounter problems. For this schema, it is clear that many activities involve financing, are located in some particular place and employ persons (e.g., a sports event or a newspaper publisher). What makes a construction project distinguishable is that it involves all those relations, plus it is made up of digging, carrying and pushing activities. So to infer a construction project's existence, we must establish the existence of one of those construction-project-unique sub-activities.

We must therefore search the *DIK* for any graphs containing the concepts *DIGGING*, *CARRYING*, and *PUSHING*. We find a schema for *PUSHING*, so that *PUSHING* can be inferred if we find some *ANIMATE* entity operating upon something. We see that the database does not contain any concepts that are sub-types of *animate*, so *PUSHING* cannot be inferred using the schema.

As there are no other graphs in the *DIK* containing any of the sub-activities we seek, we could then look at the type hierarchy, which is not reproduced in this paper. While some of the activities (e.g. *DIGGING*, *CARRYING*, and *PUSHING*) are sub-types of *ACT*, this is of limited value since assertions about *ACTs* are not necessarily true for all of its sub-types. We assume that none of the activities has any sub-types of their own to spur further searches in the *DIK*, so there is no further *DIK* knowledge that can be used.

We now scan the *DSK* for any appearance of the three activities. We see that all three of them appear, associated with various concepts: a *BLADE* is used for *PUSHING*, a *BUCKET* for *DIGGING* and a *LOADER* for either *CARRYING* or *PUSHING*. None of these concepts appears in the specification graph.

Can any of the concepts BLADE, BUCKET or LOADER be inferred from the *DBSG*? We scan the *DIK* for any occurrence of these concepts, and there are none. We scan the *DSK* for them, and find each of them included in a part relation. BLADE is part of a TRACTOR, BUCKET is part of a BACKHOE, and LOADER is part of a BACKHOE. By the general inference rule in the *DIK*, we can therefore infer the existence of a part if we can infer the existence of its whole. For any instance of BACKHOE and TRACTOR in the database, we can infer, by a direct path, the existence of a construction project. If the database includes an order for a BACKHOE from Bridgeport, California, then we can infer that a construction project exist at Bridgeport.

4. STATUS AND CONCLUSIONS

This research is just getting under way at the University of Alabama in Huntsville. The research rests firmly on previous inference and conceptual graph work of the authors. The initial work performed on this research is the development of the AERIE model, associated examples and the initial work on designing the Inference Analysis Tool. We believe that this model is unique in its use of conceptual graphs to describe both the semantics of the database as well as the common and domain-specific knowledge that would be available to a potential adversary.

While the AERIE Model is in an early development stage, it has proven itself useful in providing a means to classify various inference types within one model. It also has provided a means to show how various inference methods can be combined into a unified inference analysis model.

Our ultimate objective is to build on this preliminary work by completing the AERIE model, and then developing a representative database that can be used to characterize examples of the various types of inference problems. Finally, we plan to use this model as the basis for the implementation of an automated Inference Analysis Tool.

REFERENCES

- [Buczowski90] Buczowski, Leon J., "Database Inference Controller," *Database Security, III: Status and Prospects, Results of the IFIP WG. 11.3 Workshop on Database Security*, Monterey, CA, September 1989, North-Holland, 1990.
- [Cox88] Cox, L. H., "Modeling and Controlling User Inference," in *Database Security: Status and Prospects*, C.E. Landwehr, ed., North-Holland, 1988.
- [Delugach91] Delugach, Harry S., "A Multiple-Viewed Approach to Software Requirements," Ph.D. Dissertation, Department of Computer Science, University of Virginia, Charlottesville, VA, May, 1991.
- [Delugach92] Delugach, Harry S., "Specifying Multiple-Viewed Software Requirements with Conceptual Graphs," *Jour. Systems and Software*, to appear either third or fourth quarter 1992.
- [Denning82] Denning, D., *Cryptography and Data Security*, Addison-Wesley, 1982.
- [Eklund90] Eklund, Peter and Gerholz, Laurie, eds., *Proc. Fifth Annual Workshop on Conceptual Graphs*, Linkoping University, Stockholm, Sweden, Aug., 1990.
- [Feller89] Feller, Andrew and Rucker, Rob, "Using Conceptual Structure Principles for Meta-Modeling a Structured Analysis Tool IDEF0," *Proc. 4th Annual Workshop on Conceptual Structures*, Nagle, Janice A. and Nagle, Timothy E., eds., pp. 4.5, AAAI- IJCAI-89, Menlo Park, CA 94025, Aug. 20-21, 1989.
- [Garvey91] Garvey, Thomas D. and Teresa F. Lunt, "Cover Stories for Database Security," *Proc. 5th IFIP W.G. 11.3 Working Conference on Database Security*, Shepherdstown, WV, November 1991.
- [Garvey91b] Garvey, Thomas D., Teresa F. Lunt and Mark E. Stickel, "Abductive and Approximate Reasoning Models for Characterizing Inference Channels," *Proceedings 1991 Computer Security Foundations Workshop*, Franconia, New Hampshire, 1991.
- [Gray91] Gray, Linda C. and Bonnell, Ronald D., "A Comprehensive Conceptual Analysis Using ER and Conceptual Graphs," *Proc. Sixth Annual Workshop on Conceptual Graphs*, Way, Eileen C., ed., pp. 83-96, SUNY Binghamton, Binghamton, NY, July, 1991.
- [Han90] Han, Chia Yung and Cheng, Yizong, "Using Conceptual Structure Principles for Representing Knowledge of Chinese Medicine," *Proc. 5th Annual Workshop on Conceptual Structures*, Eklund, Peter and Gerholz, Laurie, eds., pp. 67-73, Linkoping University, Boston & Stockholm, 1990.
- [Hinke88] Hinke, Thomas H., "Inference Aggregation Detection in Database Management Systems," *1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.
- [Hinke90a] Hinke, Thomas H., "Database Inference Engine Design Approach," in *Database Security, II: Status and Prospects, Results of the IFIP WG. 11.3 Workshop on Database Security*, Kingston, Ontario, Canada, October 1988, C. E. Landwehr, ed., North-Holland, 1990.
- [Hinke90b] Hinke, Thomas H., Response to Research Question 3 in "Research Questions List, Answers, and Revision," (e.d., Carl E. Landwehr) in *Database Security, III: Status and Prospects, Results of the IFIP Working Group 11.3 Workshop on Database Security*, Monterey, CA, September 1989, North-Holland, 1990.

- [Hinke91] Hinke, Thomas H., "Query By Committee," Ph.D. Dissertation, Dept. of Computer Science, University of Southern California, Los Angeles, CA, Aug. 1991.
- [Lunt89] Lunt, Teresa F., "Aggregation and Inference: Facts and Fallacies," *1989 IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA, May 1989.
- [Matloff88] Matloff, N. S., "Inference Control vs. Query Restriction vs. Data Modification: A Perspective," in *Database Security: Status and Prospects*, North-Holland, C.E. Landwehr, ed., 1988.
- [Morgenstern87] Morgenstern, Matthew, "Security and Inference in Multilevel Database and Knowledge-Base Systems," *Proc. SIGMOD 1987*, Association of Computing Machinery, 1987.
- [Morgenstern88] Morgenstern, Matthew, "Controlling Logical Inference in Multilevel Database Systems," *1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.
- [Nagle89] Nagle, Janice A. and Nagle, Timothy E., eds., *Proc. Fourth Annual Workshop on Conceptual Structures*, AAAI, IJCAI-89, Detroit, MI, Aug., 1989.
- [Sowa84] Sowa, John F., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley Publ. Co., Reading, MA, 1984.
- [Su90] Su, Tzong-An and G. Ozsoyoglu, "Multivalued Dependence Inferences in Multilevel Relational Database Systems," *Database Security, III: Status and Prospects, Results of the IFIP Working Group 11.3 Workshop on Database Security*, Monterey, CA, September 1989, North-Holland, 1990.
- [Thuraisingham91] Thuraisingham, Bhavani, "The Use of Conceptual Structures for Handling the Inference Problem," *Proc. 5th IFIP Working Group 11.3 Working Conference on Database Security*, Shepherdstown, WV, November 1991.
- [Way91] Way, Eileen C., ed., *Proc. Sixth Annual Workshop on Conceptual Graphs*, SUNY Binghamton, Binghamton, NY, July, 1991.

Inference Through Secondary Path Analysis

Leonard J. Binns

Office of INFOSEC Computer Science

United States Department of Defense
Fort George G. Meade, Maryland

Abstract:

Inference, the ability to deduce classified information from unclassified information¹, is a leading security issue in the field of Multilevel Secure Database Management Systems (MLS DBMSs).

Trusted systems are designed to prevent an unauthorized flow of data. However, inference techniques gather unauthorized information in a seemingly proper manner. Standard models for controlling information flow (i.e.: Bell-LaPadula) cannot detect unauthorized access gained by utilizing inference techniques.

This paper addresses a specific class of inference attack: inference through secondary path analysis. This method of inference attack is characterized as one which seeks out alternative paths between two attributes whose primary path is classified. Our goal is to formally define this type of attack as a preamble to its solution.

1. Literally: "unauthorized information from authorized data"

Introduction

Inference is not a monolithic problem, and cannot be treated as such. There are a variety of techniques which can be used to infer unauthorized information from a database system. Consequently, no single definition of inference is adequate. Although it may be possible to derive a formal definition which encompasses every variation of the inference problem, such a broad definition is of little use. Since the general inference problem is unsolvable, any holistic solution based on such a general definition will not be practical.

The ultimate goal of inference research is to develop a comprehensive inference policy capable of being implemented. To accomplish this, it is essential to recognize each class of inference individually, and propose solutions to each of these specific classes. Only then will it be possible to bring each of these individual solutions together within the structure of a single, comprehensive inference policy.

This paper focuses on a specific class of inference referred to as inference through secondary path analysis. It is one of the classical techniques used to circumvent security where the primary means of querying the system denies access to the intended relationship. In this approach, a user attempts to find alternative ways of joining data to yield a relationship which would otherwise be denied.

This paper will:

- Formally define potential inference through secondary path analysis in a record level labeling environment; i.e. design time analysis
- Apply this definition to an element level labeling environment
- Consider *run time* analysis based on the results of *design time* analysis, and
- Expand the inference definition to consider the *likelihood* that a user could substantiate a potential inference path beyond a given threshold

Inference Through Secondary Path Analysis

In designing any relational database system the user generally has some idea of how each table will be accessed, and how some of these tables will be joined to derive a more complex relationship between attributes. Figure 1 shows an Emp-Grade relation and a Grade-Salary relation. It is likely that the user who designed this schema intended the relationship between employee and salary be derived by joining these two tables. Consider this path a primary path between employee and salary. If this relationship is sensitive, the designer may choose to protect the relationship by classifying an element of that path. Classifying the grade column, for example, prevents an unclassified user from completing the primary path.

Given a large enough database it is not likely that the designer is aware of all possible interactions between tables. Alternative, hence *secondary* paths may exist which ultimately relate two attributes in a manner unforeseen by the designer. For example, figure 2 introduces two more tables; Emp-Job lists an employee's job description, and Job-Grade specifies the minimum grade required to fill a specific job. A hostile user could determine an employee's minimum salary based on his current job description. This is graphically illustrated in figure 3. If the classification of the secondary path is different than the primary path then a classification inconsistency exists. The notion of which path is primary and which is secondary is not critical. Classification inconsistency is not acceptable regardless of which path is conceived to be "primary."

The purpose of an inference tool is to detect classification inconsistencies which arise from poor database design, and suggest a proper course of action to remedy the problem.

Emp-Grade (EG) [U-C]		Grade-Salary (GS) [U]	
Employee[U]	Grade[C]	Grade	Salary
Smith	13	2	15,000
Jones	2	13	45,000

employee (EG) grade (EGxGS) salary [C]

Figure 1

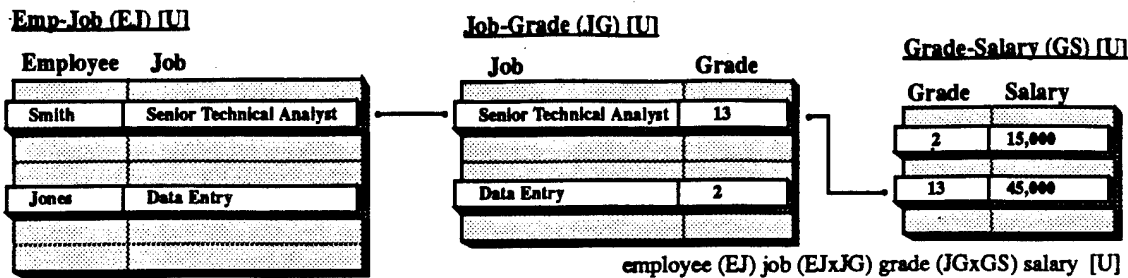


Figure 2

Increased Knowledge Leads to Improved Security

A common belief is that the more information you have to protect, the more vulnerable your system becomes. This is quite the opposite from an inference engine's point of view. Removing all classified data from a system does not prevent the unauthorized user from inferring classified information, since by definition he is only using authorized data to begin with; what it does is prevent an inference tool from being aware of any security breach. Information exists, whether we represent it in our database or not. By failing to acknowledge it does not make it any less true. In this sense, a record exists whether or not it is actually in the table. The danger arises when information is acknowledged in the database at a classification lower than the actual classification. Misclassification has two general flavors, misclassification within a record (entity) and misclassification within an aggregation of records. It is assumed that users will properly classify individual records based on some classification guideline; the burden of detecting classification inconsistencies within aggregates falls to the database's inference engine. The more world knowledge an inference tool has, the greater it's potential to discover classification inconsistencies.

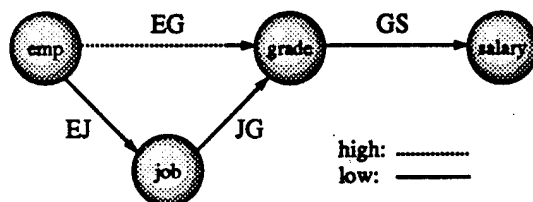


Figure 3

Defining Aggregates

According to Lin[4]¹, there is no inference between employee and salary in figures 1 and 2. By his definition, since employee and salary are each unclassified, and the relations (Emp-Job, Job-Grade, and Grade-Salary) are each unclassified, then taken together they remain unclassified. However, considering the possible intent of classification the forthcoming definition indicates inference *does* exist.

This is based on the assumption that if there exists a set of attributes $A = \{a_0, \dots, a_j, \dots, a_n\}$ and a set of relations $R = \{r_0, \dots, r_{n-1}\}$ which substantiates a path p between a_0 and a_n such that the level(a_j) in p is classified and level(a_i) in p is unclassified ($i \neq j$), then level(p) is classified. Furthermore, it is assumed that the reason a_j is classified is to protect the relationship between a_0 and a_n . If the relationship between two attributes is classified, then any information based on that relationship is classified. And so, the aggregate $\{a_0, a_n\}$ is stated to be classified even though a_0 and a_n are both unclassified.

Relating this back to figure 1, the path defined by the attributes {emp, grade, salary} is [C]. By considering the possible intent of classification we define the aggregate {emp, salary} to be [C]. Given that, we must consider the path *emp (EJ) job (JG) grade (GS) salary* to be an inference path and a violation of intended security.

Automatically defining the classification of aggregates has been addressed only informally here. This paper does not formally address the automatic classification of aggregates; however, the formal definition of inference through secondary path analysis captures the spirit of what we have discussed here.

1. Strictly speaking the schema described in figures 1 and 2 do not fit Lin's model, however it is essentially the same as the schema in Appendix A which *does* adhere to Lin's model.

Defining Inference Through Secondary Path Analysis

The definition of inference through secondary path analysis uses the notion of a path, and level of a path. These are defined here, and used in the formal definition of inference that follows.

A *path* identifies the set of attributes and relations used to substantiate a relationship between two attributes; it is a road map showing how the attributes are joined. The smallest path is between two attributes in the same relation, and by definition has a length of one. The Grade-Salary (GS) table in figure 1 shows the relationship between grade and salary. It has a length of one and the path is written *grade (GS) salary*. More complex relationships use a recursive definition for path. Each join increases the length by one. The relationship between employee and salary in figure 1 is substantiated by joining the Emp-Grade and Grade-Salary relations. The path has a length of two and is written *employee (EG) grade (EGxGS) salary*. Cyclical paths are not allowed; neither tables nor attributes can be revisited in a path.

A path of length (n=1) is defined as:

$$P(a_0, a_1, A, R) = [a_0(r_1) a_1 | \\ \wedge a_0, a_1 \in A \\ \wedge a_0 \neq a_1 \\ \wedge r_1 \in R \\ \wedge a_0, a_1 \in r_1 \\ \wedge \text{Cardinality}(A) = 2 \\ \wedge \text{Cardinality}(R) = 1]$$

A path of length (n > 1) is defined as:

$$P(a_0, a_n, A, R) = [P(a_0, a_{n-1}, A - a_n, R - r_n) (r_{n-1} \times r_n) a_n | \\ \wedge a_0, a_{n-1}, a_n \in A \\ \wedge a_0 \neq a_{n-1} \neq a_n \\ \wedge r_{n-1}, r_n \in R \\ \wedge r_{n-1} \neq r_n \\ \wedge a_{n-1}, a_n \in r_n \\ \wedge a_{n-1} \in r_{n-1}]$$

Where

A is a set of n+1 attributes, and
R is a set of n relations

The *level* of a path is defined by the relations used in traversal; it is composed of the path's hierarchical and non-hierarchical security levels unioned together. The hierarchical level is the least upper bound of all the hierarchical levels encountered in the path. The non-hierarchical level is the union of all non-hierarchical levels encountered. Although tables can be multilevel, for the purpose of inference analysis it is assumed that a table has a single security level at any point in time. For example, the Mission table in figure 4 contains unclassified and confidential missions; but no record or mission is both unclassified and confidential at the same time¹. When accessing flight C1A2946 the Mission table is effectively confidential, and when accessing flight C1A3743 it is effectively unclassified.

The level of a path *p* at time *t* is defined as:

$$L(p, t) = [L_h(R, t) \cup L_c(R, t) | \\ \exists a_0, a_n, A [p = P(a_0, a_n, A, R) \wedge t \in \text{time}]]$$

Where:

$$L_h(\text{nil}, t) = U \\ L_h(R, t) = [\text{Level}(r, t) | r \in R \wedge \text{Level}(r, t) \geq L_h(R - r, t)]$$

$$L_c(\text{nil}, t) = \text{nil} \\ L_c(R, t) = [\text{Comp}(r, t) \cup L_c(R - r, t) | r \in R]$$

Level(*r*, *t*) = Hierarchical security level
associated with relation *r* at time *t*

Comp(*r*, *t*) = Non-hierarchical compartment(s)
associated with relation *r* at time *t*

R = set of relations

{U, C, S, TS} = Hierarchical security levels,
and U < C < S < TS

MISSION (M)		[U-C]	
Flight#	Dest	Cargo	Mission_type
C1A2946	Iraq	Recon Scope 2000	RECON
C1A3743	Europe	Medical Supplies	SUPPLY

C
U

Figure 4

1. This applies to record level labeling. Element level labeling schemes will be addressed forthcoming.

MISSION (M)		
Flight#	Dest	Mission_type [U-C]
C1A2946	Iraq	RECON [C]
C1A3743	Europe	SUPPLY [U]

Figure 5

Potential inference through secondary path analysis exists if there are multiple paths between two attributes such that the levels are potentially inconsistent. This analysis does not examine actual tuple values; it is an analysis of the database design and should be done during the design or re-design of the database schema.

The following is a formal definition of potential inference through secondary path analysis in a record level labeling environment. Potential inference exists between two attributes a_0 and a_n if the following condition holds:

$$\begin{aligned}
 I(a_0, a_n) = [& \text{TRUE} \mid \exists A_1, A_2, R_1, R_2, p_1, p_2, t \\
 & [p_1 = P(a_0, a_n, A_1, R_1) \\
 & \wedge p_2 = P(a_0, a_n, A_2, R_2) \\
 & \wedge L(p_1, t) \neq L(p_2, t) \\
 & \wedge \forall r_1, r_2 (r_1, r_2 \in (R_1 \cap R_2) \wedge r_1 = r_2 \\
 & \quad \rightarrow \text{Level}(r_1, t) = \text{Level}(r_2, t) \\
 & \quad \wedge \text{Comp}(r_1, t) = \text{Comp}(r_2, t))]]
 \end{aligned}$$

This definition does not consider the polyinstantiation of tuples at different security levels; inference caused by polyinstantiated data is specifically addressed in [1].

The same relation can be used in both paths when proving inference, such as the GS relation in figure 3. However, if the same relation exists in both paths, it must be assigned the same security level in each for the purpose of determining inference; i.e. because a table is multilevel in and of itself cannot be cited as the cause of inference. This is consistent with the notion that although tables can be multilevel, they are effectively single level at any point in time.

Applying Definition to an Element Level Environment

The notion that a table has a single security level at any point in time, does not hold true for column or element level labeling environments. Consider the Mission table in figure 5, which has been modified from the previous example. The relationship between C1A2946 and Iraq is unclassified, while at the same time the relationship between C1A2946 and RECON is classified. Effectively, this relation *does* have two different security levels at the same point in time. In this case, for each attribute having a potentially different classification than its peers, the relationship between the attribute and its peers must be treated as a logically different relation. This is reflected in the labeling of the graph's arcs in figure 6; in a record level environment, all arcs would be labeled the same, since they represent the same table. If this concept is adhered to, the inference definition proposed for the record level labeling environment can be applied to the column and element level labeling environments.

In the SeaView approach where multilevel tables are decomposed into single level tables, this can be accomplished by treating each fragment of the decomposed table as a logically different table;¹ less those fragments whose only difference is their classification.

In either environment (record, column, or element), breaking the inference path involves reclassifying at least one table in the offending path, or redesigning the database schema.

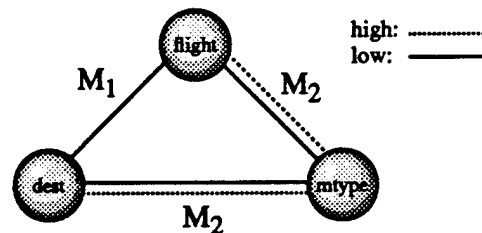


Figure 6

1. For the purpose of determining inference only

Run-Time Analysis

Removing all potential inference paths is not always feasible. It may be that forcing strict policy on a user community will cause them to abandon security for usability; or the schema may be adequate pending consistent classification of entities across various tables. In either case, we must consider run-time analysis based on the potential inference channels detected during design time.

The definition of a path yields the attributes, relations, and order of traversal used to substantiate the path. So the definition not only indicates whether a path exists between two attributes, but also indicates what that path *is*. Given the potentially offending inference path, run-time analysis would monitor its existence. Each time a relation in the primary or secondary path is modified, the run-time analysis would fire a trigger to detect if a classification inconsistency exists.

For example, consider the following two paths:

$p_1 = a_1(r_1)a_2[S]$
 $p_2 = a_1(r_2)a_3(r_2 \times r_3)a_4(r_3 \times r_4)a_2[U-S]$

The query that would substantiate p_1 is:

```
SELECT a1, a2
FROM r1;
```

The query that would substantiate p_2 is:

```
SELECT a1, a2
FROM r2, r3, r4
WHERE r2.a3 = r3.a3 and r3.a4 = r4.a4;
```

Finally, the query that would substantiate an inference between a_1 and a_2 :

```
SELECT a1, a2
FROM r1, r2, r3, r4
WHERE r2.a3 = r3.a3 and r3.a4 = r4.a4
AND r1.a1 = r2.a1
AND NOT (LUB(LEVEL(r1)) =
LUB(LEVEL(r2), LEVEL(r3), LEVEL(r4)));
```

This is the flavor of a trigger that would execute each time either r_1 , r_2 , r_3 , or r_4 were modified. If the result of the query is non-null, the inference path is substantiated. Of course, this assumes the ability to

use the level of a record within the query. The functionality of a least upper bound (LUB) operator is not critical; the query can be rewritten without its use.

Triggers that monitor the existence of an inference path must run at the level of the primary path¹, or as a trusted process. The run-time mechanism can flag the active inference channel to the high user, but cannot relay this information to the low user lest a covert channel be created. Therefore, the database administrator or high daemon running on his behalf must continue to monitor the system for inference flags.

Thuraisingham[7] has been working on constraint processing in multilevel database systems. She has proposed the concept of a constraint engine which enforces human specified constraints. Interfacing the constraint engine with an inference controller rather than the human would allow constraints to be generated automatically and more thoroughly. Based on the input from the inference controller, the constraint engine would allow the low user to access any (but not all) of the relations which make up an inference path.

We have given only a hint of how to map potential inference to run-time analysis. Although we have shown by example only, we intend to revisit this topic more thoroughly at a later date.

Quantifying the Potential of Inference

The likelihood that a potential inference path can be exploited can be described as the probability of traversing the database schemas from beginning to end of the inference path, together with the probability that the high path actually exists.

A path of length one between two attributes is recognized if the probability that the first attribute uniquely identifies the second attribute meets or exceeds the given threshold p ($0 \leq p \leq 1$). Longer paths are defined recursively so that the path is recognized if the probability of completing the entire path meets or exceeds the given threshold p ; the probability of

1. Considering that the level of the primary and secondary paths may be incomparable (due to compartments), it is more accurate to say that the trigger must run at a level which dominates both the primary and secondary paths.

completing an entire path is determined by multiplying the probabilities of each subpath.

A path of length ($n=1$) with threshold p is defined as:

$$P(a_0, a_1, A, R, p) = [a_0(r_1) a_1 | \\ \wedge a_0, a_1 \in A \\ \wedge a_0 \neq a_1 \\ \wedge r_1 \in R \\ \wedge a_0, a_1 \in r_1 \\ \wedge \text{Cardinality}(A) = 2 \\ \wedge \text{Cardinality}(R) = 1 \\ \wedge \text{Probability}(r_1(a_0, a_1)) \geq p]$$

A path of length ($n>1$) with threshold p is defined as:

$$P(a_0, a_n, A, R, p) = [P(a_0, a_{n-1}, A, R, p') (r_{n-1} \times r_n) a_n | \\ a_0, a_{n-1}, a_n \in A \\ \wedge a_0 \neq a_{n-1} \neq a_n \\ \wedge r_{n-1}, r_n \in R \\ \wedge r_{n-1} \neq r_n \\ \wedge a_{n-1}, a_n \in r_n \\ \wedge a_{n-1} \in r_{n-1} \\ \wedge p' = p / \text{Probability}(r_n(a_{n-1}, a_n))]$$

Potential inference above some threshold p ($0 \leq p \leq 1$) exists if there are multiple paths between two attributes such that the levels are not equal and the combined probability of completing each path meets or exceeds p .

$$I(a_0, a_n, p) = [\text{TRUE} | \exists A_1, A_2, R_1, R_2, p_1, p_2, t, p', p'' \\ [p_1 = P(a_0, a_n, A_1, R_1, p') \\ \wedge p_2 = P(a_0, a_n, A_2, R_2, p'') \\ \wedge p' * p'' \geq p \\ \wedge L(p_1, t) \neq L(p_2, t) \\ \wedge \forall r_1, r_2 (r_1, r_2 \in (R_1 \cap R_2) \wedge r_1 = r_2 \\ \rightarrow \text{Level}(r_1, t) = \text{Level}(r_2, t) \\ \wedge \text{Comp}(r_1, t) = \text{Comp}(r_2, t))]]]$$

Where $\text{Probability}(r(a_1, a_2))$ is defined as the probability that a_1 will uniquely identify a_2 through relation r . The method used for determining these probabilities is a subject in and of itself, and will not be addressed in the context of this paper.

Disregarding potential inferences below a certain threshold would both reduce the number of false inferences identified by the inference engine, and significantly reduce the time required to perform inference analysis. Reducing complexity is critical if we intend to provide inference analysis within an acceptable time constraint.

Considering Path Length

The probabilities mentioned heretofore do not take into account the human's ability to *recognize* potential paths. For example, the probability that a person's social security number uniquely identifies their pay grade is one, and the probability that a specific pay grade uniquely identifies the corresponding salary is one. Therefore the probability that a person's social security number can be used to uniquely determine that person's salary is one. However, the probability that all users will be aware of this path is not one¹.

Imagine a database where all subpaths had a probability of one. Still, the longer an inference path, the less likely the user will be able to substantiate that path. Given enough information, the user will be overwhelmed and less likely to complete or "see" all possible inference paths.

The reason for considering path length is again to reduce complexity. We must balance our ability to protect, against our adversary's ability to attack.

The advantage of considering path length instead of probability is the elimination of the need to correctly ascertain the proper weights for each path in the database. Not only is this a difficult burden, but one which requires continuing maintenance since the correct weights are potentially volatile.

Potential inference bounded by length l exists if there are multiple paths between two attributes such that the combined length of the primary and secondary paths is less than l .

$$I(a_0, a_n, l) = [\text{TRUE} | \exists A_1, A_2, R_1, R_2, p_1, p_2, t, l', l'' \\ [p_1 = P(a_0, a_n, A_1, R_1, l') \\ \wedge p_2 = P(a_0, a_n, A_2, R_2, l'') \\ \wedge l' + l'' \leq l \\ \wedge L(p_1, t) \neq L(p_2, t) \\ \wedge \forall r_1, r_2 (r_1, r_2 \in (R_1 \cap R_2) \wedge r_1 = r_2 \\ \rightarrow \text{Level}(r_1, t) = \text{Level}(r_2, t) \\ \wedge \text{Comp}(r_1, t) = \text{Comp}(r_2, t))]]]$$

1. Although in this simplistic example, it would be very close.

Where:

$$P(a_0, a_1, A, R, I) = [a_0 (r_1) a_1 |$$

$$\begin{aligned} & a_0, a_1 \in A \\ & \wedge a_0 \neq a_1 \\ & \wedge r_1 \in R \\ & \wedge a_0, a_1 \in r_1 \\ & \wedge \text{Cardinality}(A) = 2 \\ & \wedge \text{Cardinality}(R) = 1 \\ & \wedge 1 \leq I] \end{aligned}$$

$$P(a_0, a_n, A, R, I) = [P(a_0, a_{n-1}, A-a_n, R-r_n, I-1) (r_{n-1} \times r_n) a_n |$$

$$\begin{aligned} & a_0, a_{n-1}, a_n \in A \\ & \wedge a_0 \neq a_{n-1} \neq a_n \\ & \wedge r_{n-1}, r_n \in R \\ & \wedge r_{n-1} \neq r_n \\ & \wedge a_{n-1}, a_n \in r_n \\ & \wedge a_{n-1} \in r_{n-1}] \end{aligned}$$

It may be the case that the longer a classified path is, the less likely the reason for classification of an element in the path is to protect the relationship between the beginning and end attributes of the path. In which case, it may be better to specify the length tolerances of the primary and secondary paths individually when defining potential inference.

Of course, it is possible to combine traversal analysis based on actual probability of traversal with the notion that inference potential diminishes as path length increases.

Implementation Considerations

While this approach attempts to automatically classify aggregates based on perceived human intent, we must allow for explicit human input. If the inference engine incorrectly identifies an inference path, the human must be allowed to inform the engine that it has misinterpreted his intent and to disregard this as an inference path.

Because of the need to bound analysis time and reduce the number of false inferences identified by the engine, we have accounted for both probability and path length in our definitions. Adjusting the constraints on path length effects our scope: Minimally, the analysis will check for simple classification inconsistencies; limiting the primary path length to one will confine our search to simple cases of inference; and relaxing our restrictions on length increases both the complexity of inference we are targeting and the cost of analysis.

Summary

There is no single definition of inference which both addresses all facets of the inference problem (logical attacks, statistical attacks, world knowledge, etc.) and is capable of being implemented. Since the general inference problem is not solvable, we must bound the inference problem. The ultimate goal of this work is to develop a comprehensive inference policy capable of being implemented. Our plan is threefold. First, individually recognize and formally define each class of inference we encounter. Second, propose solutions to each of the formally defined classes. Finally, bring each of the solutions together within the structure of a single, comprehensive inference policy; recognizing those classes of inference for which we have no practical solution. To this end, we have formally defined two classes of inference: inference through polyinstantiation[1] and inference through secondary path analysis. A third class, inference through a common link, has been recognized but not yet formally defined.

Inference through secondary path analysis can be characterized as multiple paths between two attributes such that the paths' security levels are inconsistent. This paper formalizes this notion of inference in a manner which easily maps to the relational model.

Lin's approach places the burden of defining aggregates on the human. This assumes the human has sufficient awareness of the system and how it can be used/abused. The approach stated here aids the human by identifying potential aggregates and inference paths automatically. There is a trade off between the two approaches. By placing the burden on the user, implementation becomes more practical. While the latter approach is theoretically more robust, it also provides a greater challenge and risk to implement.

Our definition of inference in a record level labeling environment is applicable to the element level environment. In the case of SeaView, this is accomplished by treating each fragment of the decomposed table as a logically different table. The drawback to this is that the decomposition is driven by data input. So, much of the analysis of even potential inference channels cannot be done at database design time unless the designer has a good concept of how the system will be utilized.

The purpose of an inference tool is to detect classification inconsistencies which arise from poor database design, and suggest a proper course of action. Ideally, an inference engine should prevent the user from creating a table that would lend itself to an illegal inference path. Where elimination of potential inference is too restrictive, run-time analysis could monitor the contents of the database based on design-time analysis. In this case the tool would allow the existence of a poorly designed database, but would flag the insertion of data that would allow a potential inference path to become an active inference path. By doing analysis at design time, overhead at query time is eliminated. Run-time analysis leads to higher overhead during update, but makes design capability more flexible.

We have shown how the potential of inference can be quantified by considering the likelihood that the inference exists (probability of traversing a path successfully) and the likelihood that a user is aware of the potential inference (examining the length of the path). Although these issues were addressed separately, they can be incorporated into a single definition.

A significant number of inference tasks are underway, for this we need to develop and refine our common ground. Formally defining what we mean when we say "inference" is a step in that direction. This paper identifies one piece of the inference problem. Our goal was to bound and specify it so that we might 1) be able to follow through with a solution, and 2) verify that the solution maps to our specification.

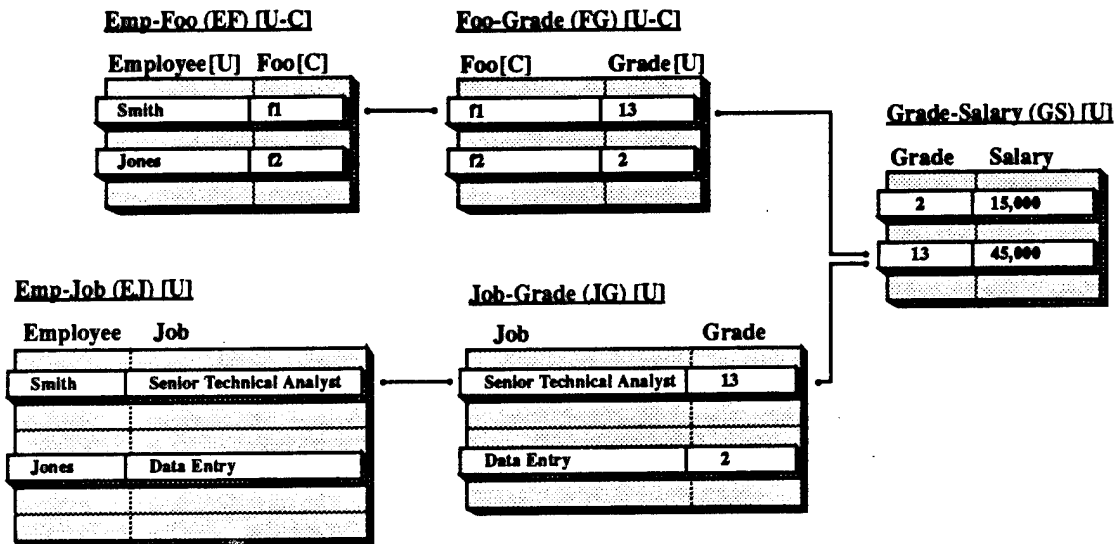
Finally, we have taken the first steps in our approach to solve inference. With each step, we're likely to realize that we've grossly underestimated the total number of steps.

References

- [1] Binns L.J. "Inference Through Polyinstantiation," *Proceedings of the Fourth RADC Database Security Workshop*, April 1991.
- [2] Garvey T.D., Lunt T.F., and Stickel M.E. "Abductive and Approximate Reasoning Models for Characterizing Inference Channels," *Proceedings of the Fourth Workshop on the Foundations of Computer Security*, Franconia, NH, June 1991.
- [3] Hinke T.H. "Inference Aggregation Detection in Database Management Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.
- [4] Lin T.Y. "Inference Free Multilevel Databases," *Proceedings of the Fourth RADC Database Security Workshop*, April 1991.
- [5] Lunt T.F. "Aggregation and Inference: Facts and Fallacies," *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1989.
- [6] Morgenstern M. "Controlling Logical Inference in Multilevel Database Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.
- [7] Thuraisingham B. "Handling Association-Based Constraints During Database Design," *Proceedings of the Fourth RADC Database Security Workshop*, April 1991.
- [8] Thuraisingham B. "The Use of Conceptual Structures for Handling the Inference Problem," *Proceedings of the Fifth IFIP WG 11.3 Working Conference on Database Security*, November 1991.

Appendix A

The example in figures 1 and 2 do not explicitly match Lin's model, but they are essentially based on the following example which does adhere to Lin's model:



This example is valid in Lin's "inference free" model. Attribute "foo" is introduced because of Lin's requirement that the values assigned to an attribute be labelled consistently throughout the database (in the previous example, value 13 for attribute "grade" existed at both the unclassified and confidential levels). Foo is being used here to classify the link between employee and grade, whereas in the previous example an instance of grade was classified to do the same thing. In essence, both examples provide the same information but do it using a different mechanism for classification. In each of these examples, the underlying reason for the classification was to protect the employee's salary. Of course, this fact is not explicitly shown in the schema -- but it can be derived by considering the possible intent of classification.

Since this example can exist in Lin's "inference free" model, and the example in figures 1 and 2 have a reasonable mapping to this example, it is argued that Lin's model should consider the schema described by figures 1 and 2 to be free of inference.

Figures 1 and 2 were used in the text instead of this example for reasons of clarity, since it is not obvious at first glance why foo exists in the schema. Foo is essentially being used here to allow the classification of a *relationship* between employee and grade, without having to classify either employee or grade.

Disclosure limitation using autocorrelated noise

George T. Duncan and Sumitra Mukherjee

The John Heinz III School of Public Policy and Management, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

Abstract

Disclosure control methods in statistical databases often rely on modifying responses to queries while approximately maintaining values of aggregate statistics. Response modification schemes suggested in the literature have adopted one of two extreme measures; the responses for repeated queries are either independent or they are totally dependent. In the former case the risk of disclosure through repeated queries is extremely high, while the latter approach suffers from the problems of increased risks under tracker attack and the possibility of a consensus on an incorrect inference. Our proposed response modification scheme based on autoregressive noise addresses each of these problems.

We have shown that under our scheme the reduction in the variance of an estimator based on repeated queries is significantly less than in the case of disclosure control methods which provide independent responses. Furthermore, the modified responses cross frequently to both sides of the true value, thus preventing a possible consensus on an incorrect inference. Most significantly, the risk of disclosure under tracker attack is significantly less under our method than when a data perturbation method is in place.

1. INTRODUCTION

Providing security to statistical databases against disclosure of confidential information is a problem both of practical concern to database administrators, and of theoretical interest to statistical and computer science researchers [5]. A single database may serve both administrative and statistical functions. For example, a medical database is used by physicians to clinically monitor individual patients; this is an administrative function since information is supplied relevant to the treatment of a particular patient. On the other hand, public health researchers require only aggregate statistics, since their goal is population inference. As defined by Adam and Wortmann [1], a statistical database

enables users to retrieve only aggregate statistics for a subset of the entities whose microdata values are represented in the database. A statistical database can be thought of as a shell for a database in which authorized users are entitled to access yielding such quantities as counts, averages, and regression coefficient estimates. They are not entitled to obtain, directly or through inference, the identity of an entity. In typical applications where sensitive information such as income, criminal history, and sexual practices may be stored, that would be a breach of confidentiality.

Direct identification is easily controlled by database software that refuses access to fields containing individual identifiers such as name and social security number. However, even when such identifiers are suppressed, it may be possible to infer confidential information about individuals based on available information. This type of disclosure is referred to as inferential disclosure, and is much more difficult to control. In their attempts to address this problem of inferential disclosure, researchers have developed several methods for inference control in statistical databases. These methods may be broadly classified into one of two classes. One class--query restriction--involves restrictions on certain types of queries, while the other class--response modification--relies on methods which modify responses to queries while approximately retaining values of aggregate statistics.

In the query restriction approach, a query may be disallowed if, for example, the number of records satisfying the conditions of a query is smaller than a specified threshold value [6,8]. The motivation behind this inference control scheme is that if very few records have certain characteristics, then these records may be easily identified. Nullifying the promise of query restriction control is the finding that through certain sequences of seemingly innocuous unrestricted queries called trackers, the answers to restricted queries can always be deduced [2].

By releasing other than the true response to a query, the response modification approach introduces uncertainty so as to reduce the risk of disclosure. A common implementation is through stochastic modification. However, under most such schemes the sequence of responses to repeated queries are stochastically independent, and hence the uncertainty may be drastically reduced by making inferences based on responses to repeated queries. Duncan and Mukherjee [4] demonstrated this problem using one such scheme, the Random Sample Query Control method discussed by Denning [3]. The variance of the estimator for the protected value was found to decrease very rapidly as the number of repeated queries increased, thus resulting in unacceptably high risks of disclosure.

The problem of variance reduction of the estimator through repeated queries may be solved by providing the same modified response every time a query is made. The database user would hence, in effect, be making queries of a modified database and gain no additional information by repeating queries. This technique of disclosure control is termed a *data perturbation method* [1]. However, a data perturbation method suffers from certain drawbacks.

In a data perturbation method, *the modified attribute is used to select records* when a

query requires selection based on the original values of the attribute. Since, in general, the distribution of the modified attribute is different from the distribution of the original attribute, the resulting set of records selected may be quite different from the set of records required by the database user. Responses to any statistical query such as count and averages may hence provide an answer to the wrong question [7].

While the problem of incorrect selection of records may be eliminated by using the original attribute values in the selection process, a second problem may arise when all users of the database obtain the same modified response to a query. For example, if the modified response is used for a statistical inference procedure such as determining a 95 percent confidence interval for the true value of the response, the interval computed by every user is identical. While in 95% of such cases the true value of the response is included in the interval, it may happen that *all users agree on a certain confidence interval which does not contain the true value of the response*. Under many circumstances a more desirable scenario is one that eliminates the possibility of such a consensus on an incorrect inference.

The above discussion suggests that under a response modification method of disclosure control, responses to repeated queries should neither be independent nor totally dependent. Under independence too much additional information is available through each repeated query, while under total dependence there is a possibility of a consensus on an incorrect statistical inference. A balance between the concern about disclosure through repeated queries and the need to prevent misleading consensus may be achieved by providing positively correlated responses to repeated queries.

In this paper we demonstrate an added advantage of providing positively correlated responses to repeated queries. We show that the precision with which values of confidential information about individual data subjects may be inferred using tools such as trackers is significantly lower when positively correlated responses are provided than when a data perturbation method is used. Since a statistical database should allow users to access only aggregate statistics and prevent the inference of information about individuals, response modification using correlated noise may be considered a more desirable disclosure control method than a data perturbation scheme.

A method for disclosure control using autocorrelated noise is presented in this paper. While our method is not vulnerable to disclosure through repeated queries as are disclosure control methods using independent responses, our scheme also addresses the drawbacks of the data perturbation method. First, record selection is based on the actual values of the attribute, rather than on the modified values. This solves the problem of providing answers to questions different from those posed by the database user. Second, the modified responses to repeated queries are distributed roughly symmetrically about the true response, thereby eliminating the possibility of a consensus on an incorrect inference. Third, under our disclosure control method, the precision with which values of confidential information may be inferred using tools such as trackers is significantly lower.

Section 2 presents the response modification scheme and shows that the reduction in variance through repeated queries is significantly less than in the case of disclosure control methods using independent responses. Section 3 demonstrates that the precision with which confidential information may be inferred using tools such as trackers is lower when correlated responses are provided than in the case when a data perturbation method is used. Section 4 investigates the distribution of the additive noise and indicates that our scheme does not suffer from the problem of possible incorrect consensus as does the data perturbation method. Implementation issues are discussed in Section 5 and our conclusions are presented in Section 6.

2. AUTOREGRESSIVE NOISE RESPONSE MODIFICATION

We propose autoregressive noise response modification (ARM). Upon sequential queries, autocorrelated noise is added to the value of an attribute before it is used to compute the response released to the user. The rationale for such a scheme is that positively correlated noise reduces the additional information content in each repeated query.

Consider a numerical attribute X for which respondent i in the database has a value X_i . For the j^{th} occurrence of a query involving this value, the modified value M_{ij} , used to compute the response is given by

$$M_{ij} = X_i + e_{ij} \quad (1)$$

where e_{ij} is the autocorrelated noise added to X_i . Specifically,

$$e_{i0} = \frac{1}{\sqrt{1-\rho^2}} \epsilon_{i0} \quad (2)$$

$$e_{ij} = \rho e_{i,j-1} + \epsilon_{ij} \quad \text{for } j=1,2,\dots, \quad (3)$$

where $\{\epsilon_{ij}\}$ is a sequence of identically distributed, independent, zero mean random variables with standard deviation σ , and $0 < \rho < 1$. It follows that for all i,j

$$E[M_{ij}|X_i] = X_i, \quad (4)$$

$$V[M_{ij}|X_i] = \frac{\sigma^2}{1-\rho^2} \quad \text{and,} \quad (5)$$

$$\text{Correlation}[M_{ij}, M_{i,j+k}] = \rho^k \quad \text{for } k=1,2,\dots. \quad (6)$$

The above properties indicate that a sequence of modified values M_{i1}, M_{i2}, \dots , is unbiased for the true value X_i of an attribute, and follows a covariance stationary process.

Now consider an aggregate query about the attribute X of a set of data subjects C for which the true response $R[C]$ is

$$R[C] = \sum_{i \in C} X_i \quad . \quad (7)$$

Under our response modification scheme, the modified response R_j^* released to the user for the j^{th} occurrence of the query is

$$R_j^*[C] = \sum_{i \in C} M_{ij} \quad . \quad (8)$$

It follows that the conditional variance of the released response is

$$V[R_j^*[C] | R[C]] = \frac{|C|\sigma^2}{1-\rho^2} \quad , \quad (9)$$

where $|C|$ is the number of data subjects satisfying the conditions of the query.

It is clear that the uncertainty about the true value of the response may be maintained at an acceptably high level by selecting appropriate values for σ and ρ . However, under repeated queries, the uncertainty about $R[C]$ may be reduced by using an estimator based on n repeated queries

$$R_n^*[C] = \frac{1}{n} \sum_{j=1}^n R_j^*[C] \quad . \quad (10)$$

It follows that

$$E[R_n^*[C]] = R[C] \quad \text{and,} \quad (11)$$

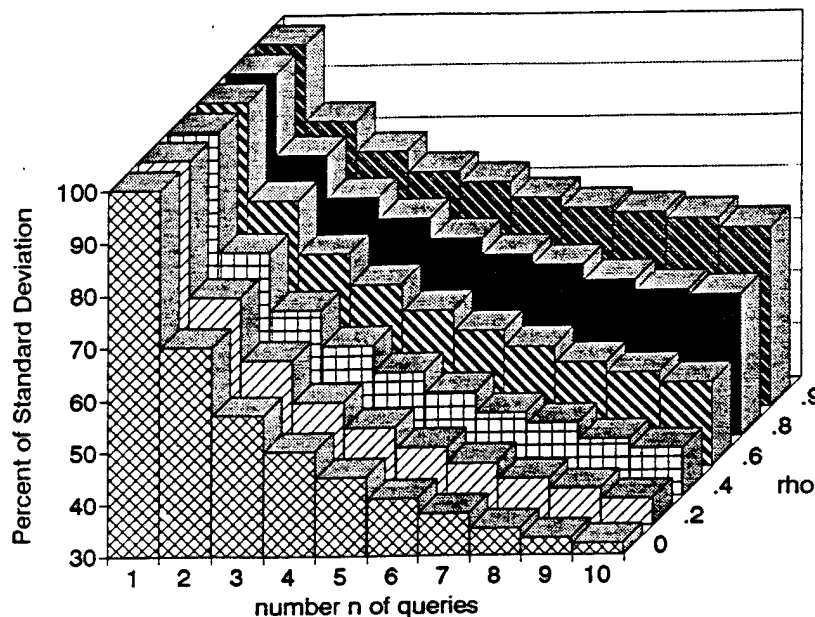
$$V[R_n^*[C]] = \frac{|C|\sigma^2}{1-\rho^2} \frac{1}{n^2(1-\rho)} \left[n - \frac{\rho(1-\rho^n)}{1-\rho} \right] \quad . \quad (12)$$

Note that in the case where responses are independent ($\rho = 0$), the variance of the estimator is given by $|C|\sigma^2/n$. As the correlation coefficient ρ increases, less and less additional information is available from each repeated query. Figure 1 shows the standard deviation of $R_n^*[C]$ as a fraction of the standard deviation of $R_1^*[C]$ for different values of ρ and n .

Notice that for a choice of $\rho = 0.9$, the standard deviation of an estimator using 6 repeated queries is about 70 percent of the standard deviation of a single response, whereas this reduction can be achieved with just 2 repeated queries in the independent ($\rho = 0$) case. Thus, in order to reduce the uncertainty appreciably, each query must be repeated numerous times. Databases containing sensitive information maintain audit trails, and such anomalous behavior as frequent repetition of queries is likely to be

discovered. This may act as a deterrent, and a database user with malicious intentions has substantially less incentive to attempt disclosure through repeated queries.

Figure 1. Effect of Autocorrelation



The risk of disclosure may be maintained within acceptable limits by ensuring that the standard deviation for the estimator is greater than some fraction λ of the standard deviation of R_j^* . This condition may be stated as

$$\frac{1}{n^2(1-\rho)} \left[n - \frac{\rho(1-\rho^n)}{1-\rho} \right] > \lambda^2 \quad (13)$$

While our proposed scheme reduces the risk of disclosure through repeated queries, the data perturbation method clearly does better since it provides no additional information through repeated queries. However, when tools such as trackers are used to infer confidential information about individuals, our method offers greater protection than does the data perturbation method. We demonstrate this advantage of our method in the next section.

3. DISCLOSURE RISK UNDER TRACKER ATTACK

Statistical databases are intended to provide aggregate statistics rather than information on individual data subjects. When a query on statistical database involves a sensitive attribute of a small group of data subjects, the query is not answered since these individuals may be easily identified. A common disclosure control method used to implement this restriction is the Query Size Restriction (QSR) control method. Under QSR control, a query is allowed only if the number of records satisfying the query is greater than some threshold value k . However, even under QSR control, answers to restricted queries may easily be inferred through a sequence of seemingly innocuous legitimate queries.

Consider, for example, the following hypothetical employee database which contains information on the salary of the employees in addition to information on their gender and designation.

Table 1. Aggregate annual Salary of employees based on their gender and designation.

Salary is represented in thousands of dollars.

The number in parenthesis shows the number of employees in that category.

	Vice President	Others	Row Totals
Male	210 (1)	2400 (59)	2610 (60)
Female	190 (1)	2000 (39)	2190 (40)
Column Totals	400 (2)	4400 (98)	4800 (100)

When QSR control is in place with $k=3$, a direct query about aggregate salary of vice presidents will not be answered since there are only two vice-presidents in the database. However, this restricted information can be obtained through the following sequence of legitimate queries:

Q1. What is the aggregate salary of employees who are vice presidents or males?

A1 = 2800

Q2. What is the aggregate salary of employees who are vice presidents or females?

A2 = 2400

Q3. What is the aggregate salary of male employees?

A3 = 2610

Q4. What is the aggregate salary of female employees?

A4 = 2190

The aggregate salary of the two vice presidents may now be computed as

$$A1+A2-A3-A4 = 400 \text{ thousand dollars.}$$

Such a sequence of legitimate queries that yield the value of a restricted attribute is called a *tracker*. QSR control may be easily subverted using trackers [2]. When the actual number of data subjects $|C|$ satisfying a query C is less than the threshold value k imposed by QSR control, the restricted value $R(C)$ may be computed as:

$$R(C) = R(C \text{ or } T) + R(C \text{ or } \sim T) - R(T) - R(\sim T)$$

where

T is a logical formula specifying a set of records such that $2k < |T|$, and $2k < |\sim T|$,

and, $\sim T$ is the complement of T .

Notice that all the four responses used to compute the restricted response are answers to legitimate queries. Trackers can always be found in any practical database and pose a significant threat to security. The risk of disclosure through trackers may be reduced by the introduction of uncertainty in the responses to queries. As discussed earlier, a common implementation of this strategy is through the addition of zero mean noise to the values of the sensitive attributes and providing masked responses to the user. When the masked responses are used in the tracker formula to estimate the response to restricted query, it can be shown that

Proposition

When disclosure control is based on noise addition, the variance of the estimator $R^*[C]$ of the restricted response under tracker attack is

$$V[R^*[C]] = \sigma^2 [|C| + 2 [N(1-\rho_1) + |C|(\rho_1-\rho_2)]] \quad (14)$$

where N is the total number of records in the database, $|C|$ is the number of records satisfying the restricted query, σ is the standard deviation of the additive noise, and ρ_i is the correlation coefficient between the additive noise for the k^{th} and the $(k+i)^{\text{th}}$ repeated query.

Proof: In the Appendix.

For the data perturbation method, the response to every repeated query is the same. That is, $\rho_i = 1$ for all i . It follows from the above proposition that the variance in the estimator for the restricted response is

$$V[R^*[C]] = \sigma^2 |C| . \quad (15)$$

Under our proposed response modification method (ARM) based on autoregressive noise $\rho_2 = \rho_1^2$. Therefore, from Equation [14], the variance of the estimator for the restricted value is given by

$$V[R^*[C]] = \sigma^2 [|C| + 2 [N(1-\rho_1) + |C|\rho_1(1-\rho_1)]] \quad (16)$$

Thus the ratio of the variance of the estimator for a restricted value under ARM to the variance of the estimator under the corresponding data perturbation method (DPM) is

$$\frac{V[R^*[C]]|_{ARM}}{V[R^*[C]]|_{DPM}} = 1 + \frac{2N(1-\rho_1)}{|C|} + 2\rho_1(1-\rho_1) > 1 \quad (17)$$

For a database to be of any practical use to the legitimate users, the restricted range ($|C| < k$, $|C| > N-k$) under QSR control must be only a very small fraction of the total number of records in the database; typically k/N is less than 1%. Consider the case where $N=100$, $|C|=2 < k$, and $\rho=.9$ (note that $k/N > 2\%$ here). Under these circumstances, the variance of the estimator under ARM is more than 11 times greater than the variance of the estimator under DPM. Notice that N/k is the dominant factor deciding the ratio of the variances, and hence under more typical circumstances ARM performs even better. For DPM to offer the same level of protection as the ARM does, the variance of the additive noise must be significantly increased. This, of course, would adversely affect the interest of the legitimate database user.

An advantage that DPM has over ARM is that no reduction in the variance of the estimator is possible through repeated queries under DPM. Hence we need to consider the effect of repeated queries in the comparison between the two methods. Under ARM, the reduction in variance through n repeated queries has been computed in the previous section. For the set of parameters under consideration ($N=100$, $|C|=2$, and $\rho=.9$) it can be shown (using [13]) that under ARM, each query must be repeated more than 100 times for the user to obtain the restricted information with the same degree of precision as under the data perturbation method.

Clearly, for the same level of noise addition, our proposed method using autoregressive noise performs better than the data perturbation method under tracker attack. This is because under DPM the noise components of the various terms in the tracker formula cancel each other out. As was noted earlier, a second problem with this constant noise based scheme is that it may lead to a consensus on an incorrect inference. A similar concern may be raised about ARM. If the additive noise component over consecutive queries remain relatively constant due to the positive nature of the correlation then ARM may suffer from the same problem as does DPM. This issue is investigated in the following section.

4. DISTRIBUTION OF MODIFIED RESPONSES OVER CONSECUTIVE QUERIES

In order to guarantee that modified responses are not repeatedly above or below the true value, we would like to ensure that the added noise e_1, e_2, \dots , do not all have the same sign for any long sequence of repeated queries. This condition may be formally stated as follows

$$E[C_N] > \gamma N, \quad (18)$$

where C_N is the number of zero crossings of the process e_1, \dots, e_N , and γ is some acceptably high fraction. This count C_N of zero crossings can be expressed as

$$C_N = \sum Z_i \quad (19)$$

where $Z_i = 1$ if $\{e_i > 0 \text{ and } e_{i+1} < 0\}$ or $\{e_i < 0 \text{ and } e_{i+1} > 0\}$
 $Z_i = 0$ otherwise.

Due to the stationary nature of the process,

$$E[C_N] = \sum E[Z_i] = N E[Z_i]. \quad (20)$$

Further,

$$\begin{aligned} E[Z_i] &= P\{e_i > 0 \text{ and } e_{i+1} < 0\} + P\{e_i < 0 \text{ and } e_{i+1} > 0\} \\ &= 2P\{e_i > 0 \text{ and } e_{i+1} < 0\} \\ &= 2P\{e_{i+1} < 0 \mid e_i > 0\} P\{e_i > 0\} \\ &= 2 \int_0^\infty \Phi(-\rho y / \sigma) \phi(y(1-\rho^2)^{1/2} / \sigma) dy \end{aligned} \quad (21)$$

where Φ and ϕ are the distribution function and density function, respectively, of the standard normal variable.

Figure 2 presents the decrease in the expected number of zero crossings with increasing values of the correlation coefficient in a sequence of 1000 responses, computed in accordance with [20] and [21].

The values for $E[C_N]$ in Figure 2 were obtained with $\sigma = 1$. In order to corroborate our theoretical results, we simulated autoregressive errors and computed the number of zero crossings in sequences of length 1000. With $\sigma = 1$ and four different values of ρ (0.6, 0.7, 0.8, and 0.9) we estimated the mean number of crossings in each case from 60 independent runs. Table 2 presents the simulation results along with our theoretically computed figures.

Table 2. Simulation Results Compared with Theoretical Results.

Correlation Coefficient ρ	Simulation Result (C_{1000})	Theoretical Result ($E[C_{1000}]$)
0.6	296	299
0.7	253	256
0.8	207	208
0.9	143	146

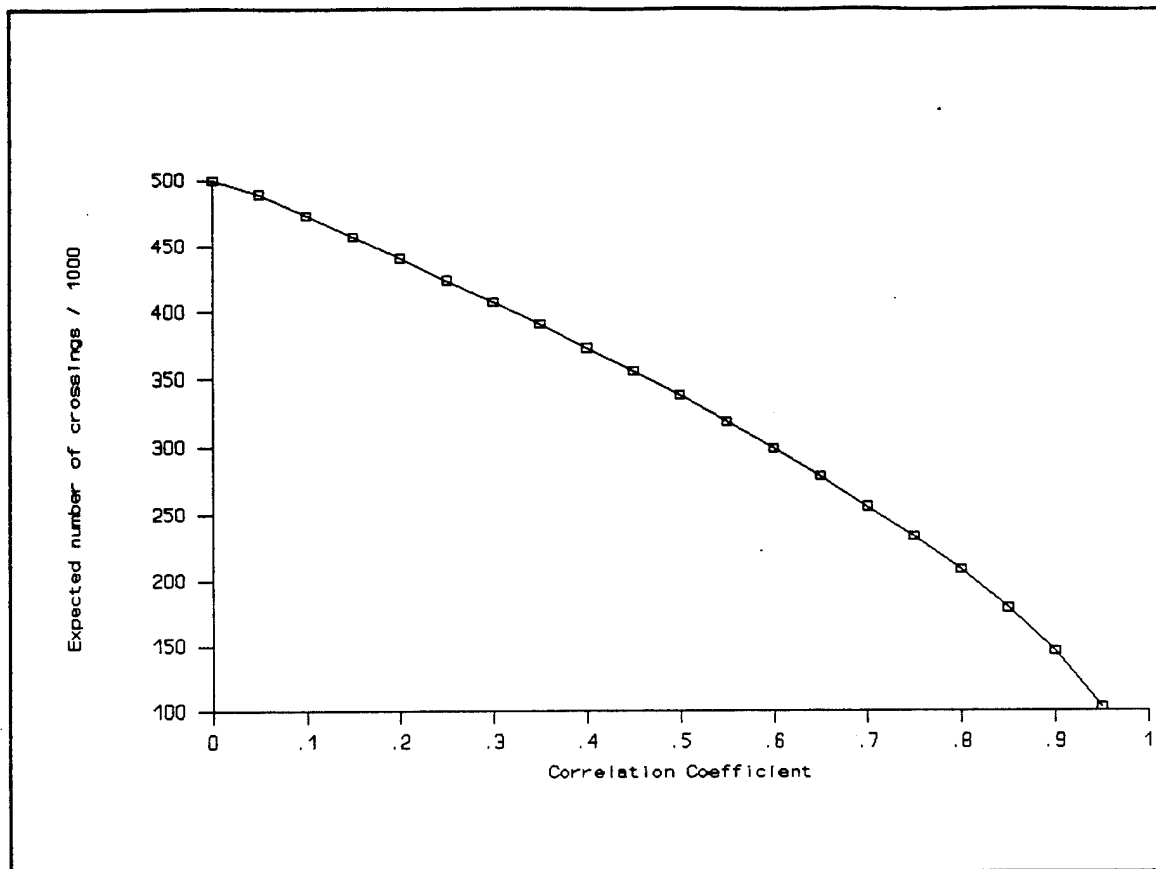


Figure 3. Expected number of zero crossings as a function of the correlation coefficient.

As evident from Table 2, the simulation results agree well with the number of crossings predicted by our theoretical results ($\chi^2 = 7.90$, $p\text{-value} = .095$). More consequentially, even with the value of the correlation coefficient as high as 0.9, the additive noise changes signs more than 140 times on average, in a sequence of length 1000. Hence, one can expect the modified responses to give values distributed on both sides of the true value for any sequence of seven consecutive responses.

To summarize this section, while the addition of autocorrelated noise serves to reduce the risk of disclosure through repeated queries, it does not suffer from the problem of possible consensus on an incorrect inference, as does the data perturbation method. Even for highly correlated noise, the sign of the noise changes frequently. It remains to be seen, however, if this additional benefit can justify the cost of implementing autocorrelated response modification. This issue is addressed in the next section.

5. IMPLEMENTATION ISSUES

We consider the cost of implementing autocorrelated response modification and compare it with the cost of data perturbation. For the purpose of our analysis, implementation cost is split into storage costs and computational costs.

In DPM, noise is added to all sensitive attributes, and these modified values are maintained in addition to the original values of the attribute. Statistical users are allowed to access only the modified values for these attributes. Computation is hence a one-time effort while the storage requirements increase by the proportion of disk space occupied by sensitive attributes.

The storage requirement for ARM is the same as that for DPM. This is because the noise added follows an autoregressive process of order one. Only the current values of the noise components need to be maintained, along with the original attribute values, in order to generate modified responses.

In terms of computation, though, the response attribute values need to be generated for each repeated query. The computation involves three basic operations: (1) independent noise generation (ϵ_i), (2) multiplication of the current value e_i of the noise by the correlation coefficient ρ , and (3) addition of the autocorrelated noise to the original value of the attribute to obtain $M_i = X + \rho e_i + \epsilon_i$.

As a fraction of the total query processing effort required, this computational load is not weighty. This is especially true for large databases, and hence the additional computation may well be justified. Moreover, in a dynamic database where confidential attributes are updated almost as often as they are queried, the computational cost of ARM is not significantly greater than the computational cost of DPM. This is because every time an attribute changes its value, even under DPM, its modified value has to be recomputed.

6. CONCLUSIONS

Response modification schemes suggested in the literature have adopted one of two extreme measures; the responses for repeated queries are either independent or totally dependent. In the former case the risk of disclosure through repeated queries is extremely high, while the latter approach suffers from the problems of increased risks under tracker attack and the possibility of an incorrect consensus. Our proposed response modification scheme, ARM, based on autoregressive noise addresses each of these problems.

We have shown that under our scheme the reduction in the variance of an estimator based on repeated queries is significantly less than in the case of disclosure control methods which provide independent responses. Furthermore, the modified responses

cross frequently to both sides of the true value, thus preventing a possible consensus on an incorrect inference. Most significantly, the risk of disclosure under tracker attack is significantly less under our method than when a data perturbation method (DPM) is in place.

7. APPENDIX

Proof of Proposition

Define

- e_{1j} : error term added to $R[C \& T]$ for the j^{th} repeated query.
- e_{2j} : error term added to $R[\sim C \& T]$ for the j^{th} repeated query.
- e_{3j} : error term added to $R[C \& \sim T]$ for the j^{th} repeated query.
- e_{4j} : error term added to $R[\sim C \& \sim T]$ for the j^{th} repeated query.

Note that

$$E[e_{ij}e_{kl}] = \begin{cases} \rho_{|j-1|} & \text{for } i=k, \\ 0 & \text{otherwise.} \end{cases}$$

Required

$$\begin{aligned} V[R^*[C]] &= V[e[C \text{ or } T]] + V[e[C \text{ or } \sim T]] + V[e[T]] + V[e[\sim T]] + \\ &\quad 2\{ \text{Cov}[e[C \text{ or } T], e[C \text{ or } \sim T]] - \text{Cov}[e[C \text{ or } T], e[T]] - \\ &\quad \text{Cov}[e[C \text{ or } T], e[\sim T]] - \text{Cov}[e[C \text{ or } \sim T], e[T]] - \\ &\quad \text{Cov}[e[C \text{ or } \sim T], e[\sim T]] + \text{Cov}[e[T], e[\sim T]] \} \\ &= \sigma^2 (|C \text{ or } T| + |C \text{ or } \sim T| + |T| + |\sim T| + \\ &\quad 2\{ E[e[C \text{ or } T] e[C \text{ or } \sim T]] - E[e[C \text{ or } T] e[T]] - \\ &\quad E[e[C \text{ or } T] e[\sim T]] - E[e[C \text{ or } \sim T] e[T]] - \\ &\quad E[e[C \text{ or } \sim T] e[\sim T]] + E[e[T] e[\sim T]] \}) \\ &= \sigma^2 (2N + |C|) + \\ &\quad 2\sigma^2 \{ E[(e_{11}+e_{21}+e_{31})(e_{12}+e_{32}+e_{41})] - E[(e_{11}+e_{21}+e_{31})(e_{13}+e_{22})] - \\ &\quad E[(e_{11}+e_{21}+e_{31})(e_{33}+e_{42})] - E[(e_{12}+e_{32}+e_{41})(e_{13}+e_{22})] - \\ &\quad E[(e_{12}+e_{32}+e_{41})(e_{33}+e_{42})] + E[(e_{13}+e_{22})(e_{33}+e_{42})] \} \\ &= \sigma^2 [|C| + 2[N(1-\rho_1) + |C|(\rho_1-\rho_2)]] \end{aligned}$$

Proved.

8. REFERENCES

- [1] Adam N.R. and Wortmann J.C. (1989), Security-Control Methods for Statistical Databases: A Comparative Study, ACM Computing Surveys, Vol 21, No.4, Dec 1989, 515-556.
- [2] Denning D.E., Denning P.J. and Schwartz M.D. (1978). The tracker: A threat to statistical database security. ACM Trans. Database Systems., 4, 1, (Mar.), 7-18.
- [3] Denning D.E. (1980) Secure statistical databases with random sample queries. ACM Trans. on Database Systems. 5, 3, (Sept.) 291-315.
- [4] Duncan G.T. and Mukherjee S. (1991). Microdata disclosure limitation in statistical databases: Query Size Restriction Control and Random Sample Query Control. Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 20-22, 278-287.
- [5] Duncan G.T. and Pearson R.W. (1991). Enhancing access to data while protecting confidentiality: prospects for the future. Statistical Science, 6, 3, 219-239.
- [6] Fellegi I.P. (1972) On the question of statistical confidentiality. Journal of the American Statistical Association, 67, 7-18.
- [7] Matloff N.S. (1986). Another look at the use of noise addition for database security. Proceedings of the 1986 IEEE Computer Society Symposium on Research in Security and Privacy, 173-180.
- [8] Schlorer J. (1975), Identification and retrieval of personal records from a statistical data bank. Methods Info. Med. 15, 1, 7-13.

Recovery management for multilevel secure database systems

Iwen E. Kang and Thomas F. Keefe
Dept. of Electrical and Computer Engineering
The Pennsylvania State University
University Park, PA 16802

Abstract- Transactions are vital for database management systems because they provide transparency to concurrency and failure. For this reason, concurrency control and recovery are important issues in multilevel secure transaction processing systems. This paper examines the security properties of database recovery management protocols. We adopt an *analytical approach* to the problem in the sense that given a system described by a protocol, we attempt to *determine* if it is secure, rather than show how the system could be constructed from secure components. This is essential because a protocol that is inherently insecure can have no secure implementation. We present a model for transaction processing systems and a corresponding security property based on noninterference and demonstrate that the property is *composable*. This allows us to consider the security of each subsystem in the transaction processing system independently. We also present a recovery protocol for multiversion schedulers and show that this protocol is both correct and secure. The behavior of the recovery protocol depends only on previous actions of the same transaction. For this reason, we believe an untrusted implementation of the recovery manager may be feasible.

1. INTRODUCTION

Multilevel Secure (MLS) computer systems provide strong mechanisms for controlling the disclosure of sensitive information. MLS database management systems (MLS DBMSs) apply the access controls of multilevel secure computers to database management systems. The goal of such systems is to protect sensitive information from unauthorized users.

Transactions are vital in database management systems because they provide transparency to concurrency and failure. Because of this, transaction processing in MLS DBMSs has received much attention recently [SHOC89], [OBRI90], [KEEF90b], [JAJ090], [GREE91], [COST91]. There are two main issues in transaction processing: concurrency control and recovery. Most of the previous work has focused on scheduling protocols that avoid illegal information flow through covert channels, however, secure recovery protocols have not yet been addressed in detail.

We distinguish two approaches to the design of a secure subsystem, *synthesis* and *analysis*. In the synthetic approach we construct the subsystem from components known to be secure and composable. An example of such a component is an untrusted subject executing on a *secure* operating system. If we can demonstrate how to construct a subsystem in this way, we know that the subsystem is no less secure than the system it is layered upon. For example, Two-Phase Locking protocol for database scheduler is shown to be insecure in [KEEF92]. This

This work has been supported by the Department of Defense under contract number MDA904-91-7043.

implies that it is impossible to construct a secure two-phase locking scheduler from untrusted subjects (or in any other way).

In the analytical approach, we attempt to *determine* if a given system is secure. Note that this does not rule out an untrusted implementation. We can describe properties of the system which imply security, e.g., the MLS property [HAIG87a, GOGU82]. This approach was taken in [KEEF92] where noninterference was applied to database transaction schedulers. A problem associated with this approach is *composition*. The composition of two trusted subsystems may lead to insecurities which do not exist in either one in isolation [MCCU90]. Analyzing the security of one system without considering the system it is incorporated into leaves us open to this problem. One solution is to completely reverify the security of the two systems together. This will require much effort. Another approach is to show that each system individually satisfies some composable security policy such as *restrictiveness* [MCCU90]. Thus, the composed system will also have this property. The composition of systems can in some cases result in nondeterminism. Demonstrating the security of nondeterministic systems is still an open problem [WITT90].

Secure recovery is also considered in [SHOC89], [OBRI90] and [GREE91]. These papers all follow a synthesis approach, each showing how to carry out recovery using untrusted subject. They also assume that subjects cannot "write-up."¹ However, this approach is not suitable for demonstrating that a protocol is not secure. In this paper we take an analytical approach, and we do allow for the possibility of writing up. We first define a model for a Transaction Processing System (TPS) and a corresponding MLS property. Following this we show that this MLS property is composable. This allows us to consider the security of each subsystem in the TPS independently.

The rest of the paper is organized as follows: Section 2 introduces the notion of multilevel security. Section 3 describes a model for a transaction processing system. Section 4 describes a security property applicable to our TPS model based on noninterference. Following this, in Section 5 we consider the problem of composition in the context of our TPS model. Section 6 briefly examines the security properties of various recovery protocols and introduces a protocol which we show to be both secure and correct. Finally, Section 7 presents our conclusion.

2. MULTILEVEL SECURITY

In this work we consider only mandatory security. Mandatory security is based on a set of security classifications partially ordered by the \geq relation. When two security classifications l_1 and l_2 satisfy $l_1 \geq l_2$ we say l_1 *dominates* l_2 . If $l_1 \geq l_2$ and $l_1 \neq l_2$ we say that l_1 *strictly dominates* l_2 , or $l_1 > l_2$. If neither one dominates the other, we say l_1 and l_2 are *incomparable* classes.

Elements of information in the system are assigned security classifications called *sensitivity levels* which represent their sensitivity. Users are assigned security classifications called *clearance levels* which represent the levels to which they are trusted. The security policy [DOD85] requires that the system satisfy the following properties [BELL76]:

¹By "write-up" we mean a transaction with classification level l writes an object with classification level l' such that $l' > l$.

Simple Security Condition

A subject may have read access to an object only if the subject's classification level dominates the object's sensitivity level.

*-Property (Star Property)

A subject may have write access to an object only if the object's sensitivity level dominates the subject's classification level.

We must also consider information flow through covert channels. A covert channel allows information to be transferred in violation of the security policy (i.e., either from a high-level subject to a low-level subject or between two subjects with incomparable security levels). Covert channels are associated with a shared resource and can be categorized as either storage or timing channels. Timing channels require that a subject or user have the ability to measure time, while storage channels do not. At higher levels of assurance, a limit is placed on the maximum bandwidth a covert channel may have [DOD85].

3. A MODEL FOR TRANSACTION PROCESSING

There are two tasks involved in transaction processing: scheduling and data management [BERN87]. The scheduler orders the actions of several concurrently executing transactions so as to maintain correctness, i.e., serializability [PAPA86]. The data manager processes the scheduled database access requests (read, write, abort and commit actions) in a way which allows recovery from system or media failure, i.e., reliability. The relationship between the components is shown in Figure 1.

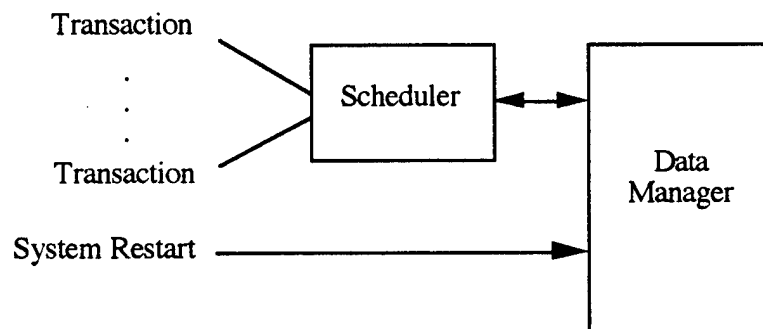


Figure 1 - Components of a Transaction Processor

Transactions issue read, write, commit and abort requests to the scheduler. The scheduler orders these requests and submits them to the data manager for execution. The data manager receives requests from the scheduler and returns results and acknowledgments. The data manager must follow an update protocol which allows proper recovery in the case of system or media failure. In this paper we consider only system failures.

Following a system failure, a system restart signal is sent to the data manager which causes the data manager to start the recovery process. To be correct, recovery should leave the database in the same state as the committed projection of the execution which precedes the failure.

The scheduler and data manager are not independent. To allow proper recovery, the scheduler must output only *recoverable* schedules [BERN87]. A schedule is recoverable if whenever transaction T_i reads from transaction T_j , T_j commits before T_i .

The design of a data manager can be simplified if the scheduler outputs only *strict* schedules. A schedule is strict if no data item is read or overwritten by another transaction until the writing transaction aborts or commits [BERN87]. This allows the effect of an aborted transaction T_i to be undone by replacing the value of every element written by T_i with the value it had just prior to the write.

The Data Manager can be decomposed into Recovery Manager, Cache Manager and Disk Manager. In the following section we describe interface, characteristics and correctness requirements of each.

3.1. Scheduler

A schedule (i.e., the output of the scheduler) is correct if it is equivalent to a *serial* schedule. A serial schedule executes a set of transactions with no interleaving (i.e., each transaction runs to completion before the next one begins). A serial schedule limits the concurrency among transactions as each transaction must wait until all previous transactions have completed. We give an example of a serial schedule below:

$T_1(S): \quad R(x, U) \quad W(y, S) \quad C$

$T_2(U): \quad \quad \quad R(z, U) \quad W(x, U) \quad C$

Here, $R(x, U)$ denotes a Read action accessing an element x which has a sensitivity level U (**Unclassified**). Similarly, $W(y, S)$ is a Write action. The classification level of the subject making the request is associated with the transaction name. For example, the action $R(x, U)$ is part of transaction T_1 which is executed by a subject with a classification level S (**Secret**). The action $T_1 : C$ is the *commit step*. It indicates the transaction's readiness to commit. However, a transaction is not committed until its commit step is executed and acknowledged by the data manager. The schedule shows the order in which actions are executed (i.e., actions on the left are executed before those on the right). All of the steps of transaction T_1 are executed before any of the steps of transaction T_2 .

Another important property of schedules is *concurrency*. A schedule is concurrent if it is an interleaved sequence of actions from two or more transactions. A serial scheduler (i.e., one which only outputs serial schedules) is correct by definition, but is inefficient. It is also insecure, since it may force low-level subjects to wait for the transactions of high-level subjects to complete.

The scheduler must produce serializable schedules and submit actions to the data manager in such a way that conflicting operations are never in the data manager at the same time. Two operations conflict if they access the same data element and at least one of them is a write.¹ We assume *handshaking* is employed to enforce the execution order of operations. If a module requires that two operations be executed in a particular order, the module must submit the first operation, wait for the executing module to acknowledge its completion and then submit the second operation. Therefore, the data manager can process any of its pending reads and writes concurrently and still achieve a serializable execution. The handshaking mechanism allows for

¹In the case of multiversion schedules, two actions conflict only if they access the same version.

the concurrent execution of multiple operations within a module. However, care must still be taken to synchronize access to shared data structures local to the module.

3.2. Recovery manager

During the normal mode of operation, the recovery manager controls the movement of data between the cache and stable storage and collects recovery information in the form of a log to aid in recovery from system failure. In the failure mode of operation, the system *restart* process must be able to utilize this recovery information (maintained on stable storage) to restore the database to a consistent state. There are three types of failures: transaction failure, system failure and media failure. A transaction failure corresponds to a transaction being aborted. A system failure results in the loss of volatile storage with the stable storage left intact. A media failure results in stable storage being corrupted. In this paper we only consider transaction and system failures. We use *stable database* to mean the state of the database in stable storage (e.g. on disk).

The recovery manager manipulates data as follows. On receiving an access request, it sends a *fix* operator to the cache manager (or a *fix_new* operator if a new object is to be created). As a result, the cache manager "pins" a cache slot for this object and returns the slot number to the recovery manager. Thereafter, the recovery manager can *read* or *write* the object cached in the slot. Pinning a cache slot insures that the cache manager will not replace it. After the access request has been acknowledged, the recovery manager can issue an *unfix* operation to allow the cache manager to replace the slot. In addition to this data manipulation, the recovery manager also maintains a log in stable storage to allow correct recovery after system failures.

The Recovery Manager must keep sufficient information in stable storage for the *restart* process to undo updates by aborted transactions and redo updates by committed ones. These requirements are called the Undo and Redo rule [BERN87]:

Undo Rule: If x's location in the stable database presently contains the last committed value of x, then that value must be saved in stable storage before being overwritten in the stable database by an uncommitted value.

Redo Rule: Before a transaction can commit, the value it wrote for each data item must be in stable storage.

In [BERN87] recovery protocols are categorized as (1) Undo/Redo, (2) Undo/No-Redo, (3) No-Undo/Redo and (4) No-Undo/No-Redo. This classification is based on characteristics of the system *restart* process, i.e., whether it requires undo or redo for handling system failure. A protocol *requires undo* if it allows an image written by an uncommitted transaction to replace a part of the *stable database*. A protocol *requires redo* if it allows a transaction to commit before all its updates are incorporated into the *stable database*.

To allow proper recovery after system failure, at times the recovery manager must insure that certain data elements are in stable storage. For example, when a transaction commits, the recovery manager may instruct the cache manager to flush the data elements written by the transaction to stable storage via the *flush* command. When a transaction aborts, the recovery manager may need to remove the effects of the aborted transaction from the stable database.

3.3. Cache Manager

The cache manager is designed to minimize the traffic between the cache and stable storage. The behavior of the cache manager can be described by its allocation and replacement protocol [EFFE84]. On a *fix_new* request, the cache manager pins an available slot and returns the slot number. In executing a *fix* request, the cache manager searches the cache for the desired

object.¹ If the data object is not in the cache, the cache allocation protocol is used to choose an available cache slot for the requested object. If no slot is available, a cache replacement protocol chooses an unpinned cache slot for replacement. If the victim slot is dirty, it is flushed to the disk and then the requested object is fetched. The cache slot containing the requested object is pinned to prevent further replacement and the slot number is returned. On a *read* request, the cache manager reads the cache slot contents and returns the result to the requester. On a *write* request, a new value is written into the cache slot and an acknowledgment is returned. An *unfix* request marks the cache slot available for replacement and returns an acknowledgment.

Data is moved between the cache and disk explicitly via the *fetch* and *flush* operations. The *flush* command takes an object name as its parameter and causes the object's contents to be written to disk if it is in the cache, otherwise the *flush* has no effect. The *fetch* command causes an object to be read from the disk and written into an available cache slot.

The outputs of the cache manager are *read_disk* and *write_disk* requests to the Disk Manager and acknowledgments to the recovery manager. The inputs to the cache manager are acknowledgments from the disk manager and requests from the recovery manager.

3.4. Disk manager

The disk manager reads and writes pages from the disk. At any time, several operations may be pending. Thus, the disk manager must schedule the execution of these operations. While the scheduling could be as simple as a first come first serve, other more complex scheduling protocols are also possible and may be required to insure security [KARG91].

The inputs to the disk manager are *read_disk* and *write_disk* requests from the cache manager. The outputs are the corresponding acknowledgments.

4. A SECURITY PROPERTY FOR TRANSACTION PROCESSING SYSTEMS

We now introduce a security models for transaction processing. As in [KEEF90b], we refer to the set of schedules which satisfy the simple and *-property as Class 2-SS. Occasionally, we consider a more restrictive class of inputs we call Class 1-SS. This is the strict subset of Class 2-SS which includes an action T: W(x) only if $\text{level}(x) = \text{level}(T)$.

We can model a Transaction Processing System (TPS) as a Temporal Labelled State Machine (TLSM). A TLSM is a deterministic state machine which accepts a sequence of input events and generates a sequence of output events. An *input event* (*output event*) is an input (output) action tagged with a non negative *timestamp* representing the time it enters (leaves) the system. We assume events never coincide and thus these timestamps are unique. Each input event is labeled with the classification level of the requesting subject which issued it. Each output event is also labeled with a classification level of the receiving subject.

We define an *event* as follows.

Definition- An *event* is a triple $\text{DOMAIN} \times \text{TIME} \times \text{SC}$, where DOMAIN is a set of actions; TIME is a set of non-negative real numbers; SC is a set of security classification levels;

¹The data object is assumed to be a fixed size page.

Note that in the definition, TIME is a set of non-negative real numbers. This implies different events never coincide. We let *dom*, *time* and *level* be functions which return the DOMAIN, TIME and SC component of an event, respectively.

Given a machine with *input domain* (a set of acceptable input actions) *I* and *output domain* (a set of output actions) *O*, we use *X* to denote the set of *input events* $\{(i, t, l) \mid i \in I, t \in \text{TIME}, l \in \text{SC}\}$ and *Y* to denote the set of *output events* $\{(o, t, l) \mid o \in O, t \in \text{TIME}, l \in \text{SC}\}$ for that machine. We now define a Temporal Labelled State Machine.

Definition- A Temporal Labelled State Machine (TLSM) is a deterministic state machine $(S, s_0, I, O, \delta, \lambda)$ where,

- *S* is the set of states of the machine;
- s_0 is the initial state of the machine;
- *I* is the input domain, giving the set of acceptable input actions;
- *O* is the output domain, giving the set of acceptable output actions;
- $\delta: S \times X \rightarrow S$ is the state transition function. It maps a state and an *input event* to the next state;
- $\lambda: S \times X \rightarrow Y$ is the output function. It maps a state and an *input event* to an *output event*;

It is clear now an input or output *sequence* is a *set* of events ordered by their *timestamps* (the TIME component of events). The notation $p = aw$ means *a* is the event with the smallest timestamp in *p* with *w* containing the remaining events.

We next define the *extended output function* and *extended state transition function* for a TLSM.

Definition- Define the **extended output function** $\hat{\lambda}: S \times 2^X \rightarrow 2^Y$

$$\begin{aligned}\hat{\lambda}(s, \phi^1) &= \phi \\ \hat{\lambda}(s, aw) &= \lambda(s, a) \cup \hat{\lambda}(\delta(s, a), w)\end{aligned}$$

Definition- Define the **extended state transition function** $\hat{\delta}: S \times 2^X \rightarrow S$

$$\begin{aligned}\hat{\delta}(s, \phi) &= s \\ \hat{\delta}(s, aw) &= \hat{\delta}(\delta(s, a), w)\end{aligned}$$

In an event based system, it is convenient to describe system behavior in terms of a set of *traces* as defined below.

Definition- A *trace* of a TLSM $m = (S, s_0, I, O, \delta, \lambda)$ is of the form:

$$s_0(x_1 s_1 y_1) \dots (x_n s_n y_n)$$

¹ ϕ denotes the empty set.

where s_0 is the initial state of m , $x_i \in X$, $y_i = \lambda(s_{i-1}, x_i) \in Y$ and $s_i = \delta(s_{i-1}, x_i) \in S$ for $i = 1, \dots, n$. In addition, the timing in a trace satisfies $time(x_1) < time(y_1) \leq time(x_2) \dots \leq time(x_n) < time(y_n)$.

If an output event y_{i-1} is identical to the input event x_i (i.e., $y_{i-1} = x_i$) then we say y_{i-1} is a *feedback event*. The timing constraint implies that no other events can occur during a state transition. This may seem not applicable to systems with long input-output delays. However, we can model delays by introducing the *tick events*. A *tick event* $\tau_i = (null, time(i), \perp)$ denotes a *null* operation occurring at $time(i)$, where \perp is the classification level dominated by all others. The tick events serve as event markers to the system. Thus, a subtrace $(x_i s_i \tau_i) \dots (\tau_j s_j y_j)$ models an output y_j in response to an input x_i which is delayed for $time(y_j) - time(x_i)$ time units.

We next define the purge function.

Definition- The **purge function** $purge : SC \times 2^{(X \cup Y)} \rightarrow 2^{(X \cup Y)}$ is defined as

$$\begin{aligned} purge(l, \phi) &= \phi \\ purge(l, alw) &= a \mid purge(l, w) && \text{if } l \geq level(a) \\ &= \tau_a \mid purge(l, w) && \text{otherwise.} \end{aligned}$$

It is easy to verify that the purge function has the following properties:

$$\begin{aligned} purge(l, p) &= purge(l, purge(l, p)) && \dots \text{purge property (1)} \\ purge(l, p \cup q) &= purge(l, p) \cup purge(l, q) && \dots \text{purge property (2)} \end{aligned}$$

We now define an MLS property for our model as in [GOGU84].

Definition- A TLSM is **MLS** with respect to a class of input schedules S which is prefix closed if for every schedule p in S , and every subject classification level l in p ,

$$purge(l, \hat{\lambda}(s_0, p)) = purge(l, \hat{\lambda}(s_0, purge(l, p)))$$

In other words, a TLSM is MLS if a subject at level l cannot distinguish the outputs in the two cases even when time is taken into account. Therefore, our definition of MLS property implies the system is free of timing as well as storage channels.

This property may be quite difficult to model or verify in practice as it requires reasoning about the real-time behavior of the system. We plan to simplify the problem through approximations in our system modelling. We make the following assumptions:

1. Events are of short duration and thus never coincide.
2. The execution of an action progresses, alternating between processing and I/O. We assume that the processing time is negligible compared to the I/O and is therefore ignored.
3. Whenever an input action arrives for processing, the machine executes for some time, produces one output action and awaits an acknowledgment. While waiting for an acknowledgment, another input action can be processed. Thus, each subsystem is multi-threaded. Scheduling of the threads is completely determined by the arrival of input events and acknowledgments. Thus, the system is deterministic.

In the next section we show that this MLS property is composable.

5. DECOMPOSITION AND THE MLS PROPERTY

To simplify security analysis, we decompose the system into subsystems. The interconnection of subsystems for a TPS is shown in Figure 2. A transaction submits its operations to the "Input" of TPS and receives acknowledgments from the "Output" of TPS. Each module sends requests for service to the next module, and receives inputs from the previous module and acknowledgments from the next. Note that *restart* is not modeled as an input to the system. This is because *restart* can only be triggered by system failure. We assume no subject is able to control system failure and thus a Trojan Horse program cannot utilize the *restart* to drive a covert channel.

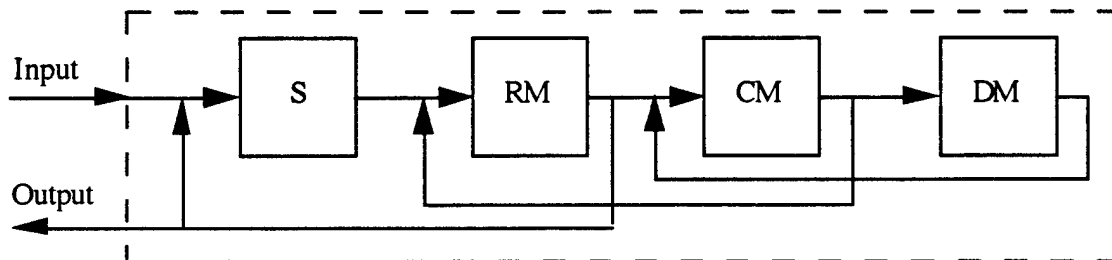


Figure 2 - Interconnection of subsystems in TPS

We will show that a TPS is secure if each of its subsystems is secure. The security requirement is that every subsystem satisfies the MLS property with respect to its set of possible inputs (Note that this includes the acknowledgments from following subsystems). We first introduce the following restriction operation.

Definition- Restriction Operator $|^A$

The restriction of a set p to the set A denoted $p|^A$ is the subset of p consisting of just those elements in p which are also in the set A .

It is easy to see that

$$\text{purge}(l, p)^{|A} = \text{purge}(l, p|^A) \quad \dots \text{purge property (3)}$$

We begin by considering a feedback configuration as shown in Figure 3. In Figure 3, a portion of the output, denoted by $Y|^x$ is fed back to m 's input. We formally define this below.

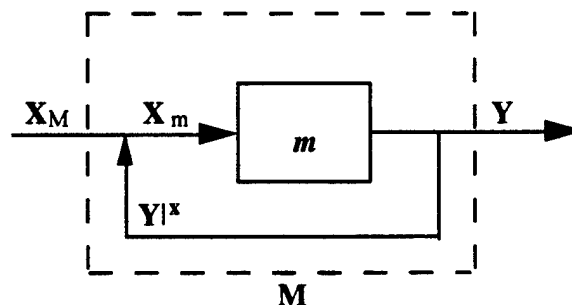


Figure 3 - A feedback configuration

Definition- Let $s_0(x_1s_1y_1)....(x_ns_ny_n)$ be a trace of a TLSM $m = (S, s_0, I, O, \delta_m, \lambda_m)$. The **Feedback Machine** of m , denoted $M = (S, s_0, (I - O) \cup \{null\}, O, \delta_M, \lambda_M)$ is also a TLSM defined as follows:

- $\delta_M(s_{i-1}, X_i) = \delta_m(s_{i-1}, y_{i-1}),$ if $dom(y_{i-1}) \in I;$
 $\delta_m(s_{i-1}, X_i),$ otherwise.
- $\lambda_M(s_{i-1}, X_i) = \lambda_m(s_{i-1}, y_{i-1}),$ if $dom(y_{i-1}) \in I;$
 $\lambda_m(s_{i-1}, X_i),$ otherwise.

From the definition it is clear that the feedback machine of a TLSM is deterministic and can be modeled as a TLSM as well.

Lemma 1- Given a trace

$$t_m = s_0(x_1s_1y_1)....(x_ns_ny_n)$$

of a TLSM m , we can find a *corresponding trace*

$$t_M = s_0(X_1s_1y_1)....(X_ns_ny_n)$$

of the feedback machine M of m such that

$$\begin{aligned} X_i &= \tau_i^1, \text{ if } y_{i-1} = x_i; \\ &= x_i, \text{ if } y_{i-1} \neq x_i; \end{aligned}$$

for $i = 1, \dots, n$.

Proof: We need to show that t_M defined this way is always a trace. We prove this by an induction over the length of t_m .

Basis. $t_m = s_0$. We have $t_M = s_0$, trivial.

Induction step. Assume Lemma 1 holds for $t_m = s_0(x_1s_1y_1)....(x_{n-1}s_{n-1}y_{n-1})$.

Let $(x_ns_ny_n)$ be the continuation of trace t_m . We have

$$\begin{aligned} \lambda_M(s_{n-1}, X_n) &= \lambda_M(s_{n-1}, \tau_n), \text{ if } y_{n-1} = x_n; \\ &\lambda_M(s_{n-1}, x_n), \text{ otherwise.} \quad \text{-by the definition of } X_n \\ &= \lambda_m(s_{n-1}, y_{n-1}), \text{ if } y_{n-1} = x_n \text{ (i.e., } dom(y_{n-1}) \in I); \\ &\lambda_m(s_{n-1}, x_n), \text{ otherwise;} \quad \text{-by the definition of } \lambda_M \\ &= \lambda_m(s_{n-1}, x_n) \\ &= y_n \quad \text{-} t_m \text{ is a trace of } m \end{aligned}$$

Similarly,

$$\delta_M(s_{n-1}, X_n) = s_n$$

In addition, t_M satisfies the timing constraint of a trace since $time(X_i) = time(x_i)$ for $i = 1, \dots, n$.

Thus, we have shown $t_M = s_0(X_1s_1y_1)....(X_ns_ny_n)$ is a trace of M . \square

${}^1\tau_i = (null, time(x_i), \perp).$

Lemma 1 implies that if p is an input sequence to m then we can find the *corresponding input sequence* q to m 's feedback machine M such that $\hat{\lambda}_m(s_0, p) = \hat{\lambda}_M(s_0, q)$. Now, consider the question: given p an input to m and q the corresponding input sequence to M the feedback machine of m . Is $\text{purge}(l, q)$ the corresponding input sequence for $\text{purge}(l, p)$? Surprisingly, the answer is **not** necessarily true.

Consider the following example: Suppose input a is a high event; feedback b ($\text{dom}(b) \neq \text{null}$) is a low event (b serves both as an input and an output); output event c is also low. Let $\lambda_m(s_0, \tau_i) = \tau_i$ and $\delta_m(s_0, \tau_i) = s_0$, for $i = a, b$; $\lambda_m(s, b) = c$ and $\delta_m(s, b) = s_b$, for any state s . Further assume $s_0(a s_1 b)(b s_b c)$ is a trace of m , then by Lemma 1, $s_0(a s_1 b)(\tau_b s_b c)$ is the corresponding trace of M . That is, $p = ab$ is the input sequence to m and $q = a \tau_b$ is the corresponding input sequence to M . Therefore, $\text{purge}(l, p) = \tau_a b$ and $\text{purge}(l, q) = \tau_a \tau_b$, where $l = \text{low}$. So, $s_0(\tau_a s_0 \tau_a)(b s_b c)$ is the trace of m corresponding to input $\text{purge}(l, p)$, and $s_0(\tau_a s_0 \tau_a)(b s_b c)$ is the corresponding trace of M by Lemma 1. However, $s_0(\tau_a s_0 \tau_a)(b s_b c)$ is not the trace corresponding to the input sequence $\text{purge}(l, q)$ of M , since $\text{purge}(l, q) = \tau_a \tau_b \neq \tau_a b$.

We will show in the next lemma that if m is MLS and q is the corresponding input sequence for p , then $\text{purge}(l, q)$ is the corresponding input sequence for $\text{purge}(l, p)$.

Lemma 2- Let M be the feedback machine of a TLSM m and q be the corresponding input sequence for p such that $\hat{\lambda}_m(s_0, p) = \hat{\lambda}_M(s_0, q)$. If m is MLS then $\text{purge}(l, q)$ is the corresponding input sequence for $\text{purge}(l, p)$ such that $\hat{\lambda}_m(s_0, \text{purge}(l, p)) = \hat{\lambda}_M(s_0, \text{purge}(l, q))$ for all classification levels l .

Proof: Let $p = x_1 x_2 \dots x_n$ and $q = X_1 X_2 \dots X_n$. Also let $\text{purge}(l, p) = x_1' x_2' \dots x_n'$ and $\text{purge}(l, q) = X_1' X_2' \dots X_n'$. Assume $t_m = s_0(x_1 s_1 y_1) \dots (x_n s_n y_n)$ is a trace of m . We know $t_M = s_0(X_1 s_1 Y_1) \dots (X_n s_n Y_n)$ is the corresponding trace of M since q is the corresponding input sequence for M . Given m is MLS and $t'_m = s_0(x_1' s_1' y_1') \dots (x_n' s_n' y_n')$ is a trace of m , we need to show $t'_M = s_0(X_1' s_1' Y_1') \dots (X_n' s_n' Y_n')$ is the corresponding trace of M (i.e., $\text{purge}(l, p)$ and $\text{purge}(l, q)$ are corresponding input sequences). Since m is MLS, we have

$$\begin{aligned} \text{purge}(l, y_1' y_2' \dots y_n') &= \text{purge}(l, \hat{\lambda}_m(s_0, \text{purge}(l, p))) && \dots \text{by def. of } t'_m \\ &= \text{purge}(l, \hat{\lambda}_m(s_0, p)) && \dots \text{by } m \text{ is MLS} \\ &= \text{purge}(l, y_1 y_2 \dots y_n) && \dots \text{by def. of } t_m \end{aligned}$$

$$\text{So, } y_i = y_i' \quad \text{if} \quad \text{level}(y_i) \leq l \text{ or } \text{level}(y_i') \leq l \text{ for } i = 1, \dots, n. \quad \dots(1)$$

Now, consider

$$(x_{i-1} s_{i-1} y_{i-1})(x_i s_i y_i) \subset t_m \text{ and } (x_{i-1}' s_{i-1}' y_{i-1}')(x_i' s_i' y_i') \subset t'_m$$

We claim

$$x_i' \neq y_{i-1}' \quad \text{if} \quad x_i \neq y_{i-1} \text{ and } \text{level}(x_i) \leq l \text{ for } i = 1, \dots, n. \quad \dots(2)$$

since if $\text{level}(x_i) \leq l$, we have $x_i' = x_i$ because $x_i' = \text{purge}(l, x_i)$. There are two cases to consider:

case 1. $\text{level}(y_{i-1}') \leq l$.

We have $y_{i-1}' = y_{i-1}$ by (1). So, $x_i' = x_i \neq y_{i-1} = y_{i-1}'$, i.e., $x_i' \neq y_{i-1}'$.

case 2. $\text{level}(y_{i-1}') \not\leq l$.

$x_i' \neq y_{i-1}'$ since $\text{level}(x_i') \leq l$.

Therefore,

$$\begin{aligned}
X_i' &= X_i, \text{ if } \text{level}(X_i) \leq l; & -\text{by } X_i' = \text{purge}(l, X_i) \\
&\tau_i, \text{ otherwise.} \\
&= x_i, \text{ if } \text{level}(x_i) \leq l \text{ and } x_i \neq y_{i-1}; & -\text{by } t_m \text{ and } t_M \text{ are corr. traces} \\
&\tau_i, \text{ otherwise.} & \text{and the application of Lemma 1} \\
&= x_i', \text{ if } \text{level}(x_i) \leq l \text{ and } x_i \neq y_{i-1}; \\
&\tau_i, \text{ otherwise;} & -\text{by } \text{purge}(l, x_i) = x_i' \text{ and} \\
& & \text{level}(x_i) \leq l \\
&= x_i', \text{ if } x_i' \neq y_{i-1}'; \\
&\tau_i, \text{ otherwise.} & -\text{by (2)}
\end{aligned}$$

By $t'_m = s_0(x_1's_1'y_1') \dots (x_n's_n'y_n')$ is a trace of m and Lemma 1, we have shown $t'_M = s_0(X_1's_1'y_1') \dots (X_n's_n'y_n')$ is the corresponding trace of M . Therefore, $\hat{\lambda}_m(s_0, \text{purge}(l, p)) = \hat{\lambda}_M(s_0, \text{purge}(l, q))$. \square

The next theorem shows a sufficient condition for MLS composability for the feedback configuration.

Theorem 1- If a TLSM m is MLS then its feedback machine M is also MLS.

Proof: Let p be an input sequence to m and q be the corresponding input sequence to m 's feedback machine M . Therefore,

$$\begin{aligned}
&\text{purge}(l, \hat{\lambda}_M(s_0, q)) \\
&= \text{purge}(l, \hat{\lambda}_m(s_0, p)) & \dots \text{by Lemma 1} \\
&= \text{purge}(l, \hat{\lambda}_m(s_0, \text{purge}(l, p))) & \dots \text{by } m \text{ is MLS} \\
&= \text{purge}(l, \hat{\lambda}_M(s_0, \text{purge}(l, q))) & \dots \text{by Lemma 2}
\end{aligned}$$

\square

We now show a sufficient condition of MLS composability for the configuration shown in Figure 4. Consider a TLSM $A = (S_A, s_{0A}, I_A, O_A, \delta_A, \lambda_A)$, and a TLSM $B = (S_B, s_{0B}, I_B, O_B, \delta_B, \lambda_B)$. Let the construction of the composite machine $m = (S_m, s_{0m}, I_m, O_m, \delta_m, \lambda_m)$ shown in Figure 4 be such that

$$\hat{\lambda}_m(s_0, p) = \hat{\lambda}_A(s_{0A}, p|^{x_A}) \cup \hat{\lambda}_B(s_{0B}, p|^{x_B})$$

where p is the input to m and $p|^{x_A}$ and $p|^{x_B}$ are p restricted to the events in the input domain of A and B , respectively. Note that if the input domains I_A and I_B are not disjoint, some input events may appear in the input of both A and B . However, the output function of A and B will map the same input event to different output events (with different *timestamps*) by our assumption that different events never coincide.

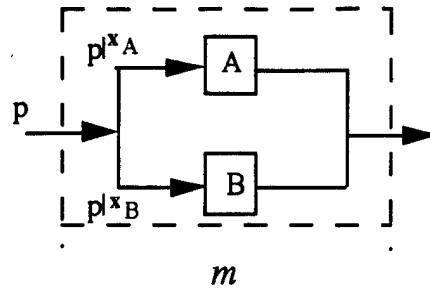


Figure 4 - Parallel configuration

Theorem 2- Consider the configuration as shown in Figure 4. If both A and B are MLS, then m is also MLS.

Proof:

$$\begin{aligned}
 & \text{purge}(l, \hat{\lambda}_m(s_{0m}, p)) \\
 = & \text{purge}(l, \hat{\lambda}_A(s_{0A}, p|^{x_A}) \cup \hat{\lambda}_B(s_{0B}, p|^{x_B})) && \dots \text{by the construction of } m \\
 = & \text{purge}(l, \hat{\lambda}_A(s_{0A}, p|^{x_A})) \cup \text{purge}(l, \hat{\lambda}_B(s_{0B}, p|^{x_B})) && \dots \text{by purge property (2)} \\
 = & \text{purge}(l, \hat{\lambda}_A(s_{0A}, \text{purge}(l, p|^{x_A}))) \cup \text{purge}(l, \hat{\lambda}_B(s_{0B}, \text{purge}(l, p|^{x_B}))) && \dots \text{by A and B are MLS} \\
 = & \text{purge}(l, \hat{\lambda}_A(s_{0A}, \text{purge}(l, p)|^{x_A}) \cup \hat{\lambda}_B(s_{0B}, \text{purge}(l, p)|^{x_B})) && \dots \text{by purge property (2) \& (3)} \\
 = & \text{purge}(l, \hat{\lambda}_m(s_{0m}, \text{purge}(l, p))) && \dots \text{by the construction of } m
 \end{aligned}$$

□

It is easy to see that the cascaded configuration in Figure 5.1 is *equivalent* to the feedback machine of m shown in Figure 6.1. By equivalent we mean that the input and output behavior are the same. The lower-case letters appeared in Figure 5-6 denote the input or output set to the corresponding TISM. Moreover, the configuration in Figure 5.2 is equivalent to machine B cascaded with the parallel configuration in Figure 6.2 (where I is the identity machine which transmits input to output). Therefore, MLS property is composable for the configuration shown in Figure 5.1 and Figure 5.2 by Theorem 1 and Theorem 2.

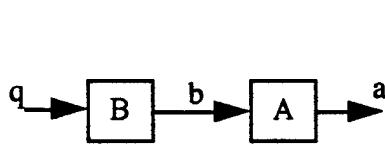


Figure 5.1- Cascaded configuration

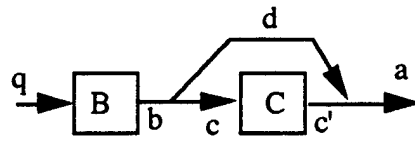


Figure 5.2- Cascaded with feed-forward configuration

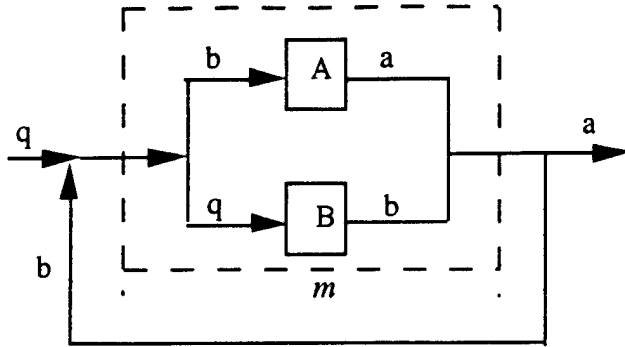


Figure 6.1- Configuration equivalent to figure 5.1

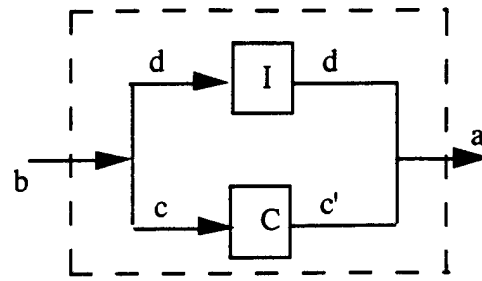


Figure 6.2- Feed-forward configuration

Now we show that if each module of a TPS is MLS, then the TPS is also MLS.

Theorem 3- The TPS shown in Figure 2 is MLS if:

1. The Scheduler is MLS; and
2. The Recovery Manager is MLS; and
3. The Cache Manager is MLS; and
4. The Disk Manager is MLS.

Proof: We will construct a TPS equivalent to the configuration shown in Figure 7. First consider only the inner most subsystem A. We use fbk_{DM} to denote the feedback (Acknowledgment) portion of the DM's output. Subsystem A without the feedback fbk_{DM} is the same as the configuration in Figure 6.1, therefore, A is MLS. By similar reasoning we can show that the composite machine M (with total output) is MLS. We then restrict the output of M to be the output (fbk_{RM}) of the TPS shown in Figure 2 which is a subset of M's original total output. If M is MLS then obviously the TPS is MLS as well. \square

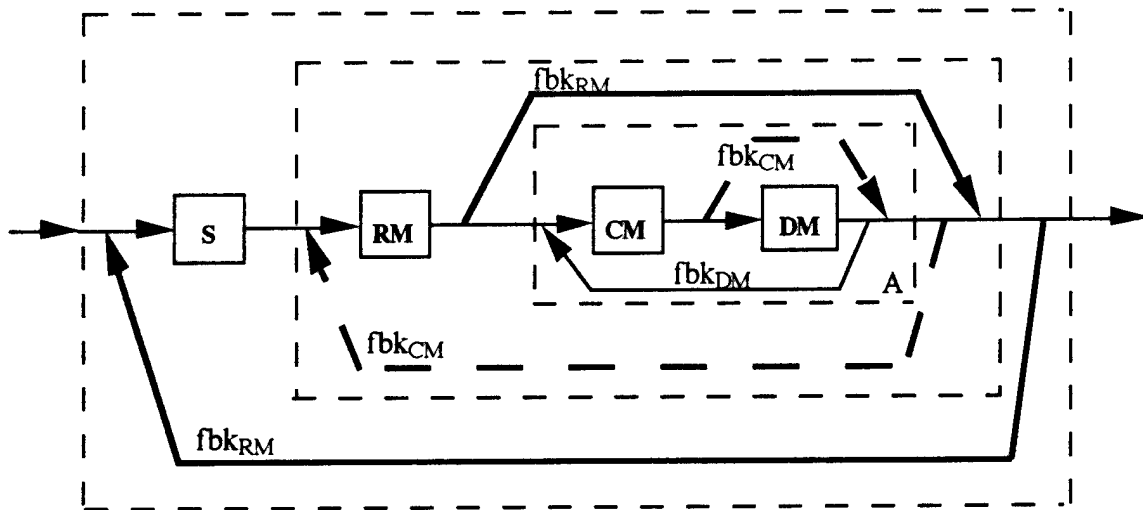


Figure 7 - The composite machine M

It is no surprise that the MLS property is composable for the TPS shown in Figure 2. In fact, using our model, any composite machine can be constructed from the basic configurations (Figure 3 and Figure 4) is deterministic. This is because of our assumption that different events never coincide and are ordered by their timestamps (the time they arrive or leave the machine).

Therefore, we can show that the MLS property is composable. This is consistent with the fact that *restrictiveness* [MCCU90] and *nondeducibility on strategy* [MILL90] are composable, since they are both equivalent to *noninterference* for deterministic systems.

Using the TLSM model we show a way to model the composition of deterministic machines. We apply this modeling technique to analyze the security of a TPS. Since scheduling and recovery are the main issues in transaction processing, we next focus on the security of recovery protocols using the MLS property. An analysis of scheduling protocols can be found in [KEEF92].

6. SECURITY PROPERTIES FOR RECOVERY PROTOCOLS

In this section we take up recovery protocols. Our goal is to *determine* whether a given protocol is secure. We do not consider how to implement the protocols using untrusted components.

We first examine the security of *strict* schedulers. Strictness is a common assumption for many recovery protocols. We then briefly analyze the security of various existing protocols. Finally, a secure recovery protocol is presented.

6.1. Strict execution

A scheduler which produces only *strict* schedules simplifies the Recovery Manager design by allowing it to roll back a transaction by simply replacing the before image of each element written by the transaction. However, *strictness* causes security problems for Class 2-SS transactions. *Strict* execution requires that whenever a transaction T_i writes to the object x , subsequent reads or writes to x from other transactions must be delayed until T_i terminates [BERN87]. If write up is allowed, the scheduler must delay low transactions who write to a high-level object x until the high-level transaction which previously wrote x terminates. Therefore, a scheduler which produces strict schedule is insecure with respect to Class 2-SS schedules. If write up is not allowed, a scheduler which is otherwise secure can be made *strict* in a secure way [KANG92].

6.2. Analysis of various recovery protocols

We distinguish two types of Recovery Managers. The first type employs Update *In-Place*. Each time a data item is overwritten, the old value is destroyed. The policy for the second type is called *Shadowing* [BERN87], each write creates a new version and old versions are not overwritten.

The Write Ahead Log protocol employs an Update In-Place policy and requires that the before image of a write be logged ahead of the write [BERN87]. However, this is sufficient only when the scheduler outputs *strict* schedules. Since a *strict* scheduler is not secure with respect to Class 2-SS transactions, this protocol is not suitable for systems which allow transactions to write up.

We can classify recovery protocols based on the time at which a transaction's updates become part of the stable database. The operation that makes a previously written page part of the stable database is called *propagation* [HAER83]. This operation writes the directory structure (if any)

for mapping data items to their locations in stable storage. We can distinguish two types of propagation strategies¹ :

Atomic: The updates of a transaction are propagated as a unit at *commit time*, such that either all or none of the updates become part of the stable database.

~Atomic: Each update is propagated as a unit to the stable database, thus, the updates of a transaction can be made visible to others even while the transaction is still active.

An example of a recovery protocol based on *Atomic propagation* is the *Shadow Page Algorithm* [BERN87]. The Recovery Manager maintains a *Master* record and two directories called *Current* and *Scratch*. The Master record stores a pointer to the Current directory. These two directories alternate between being the current and being the scratch directory, depending on the pointer in the Master record. The directories must be kept in stable storage but can be cached for efficient access. Each modification of the Master record and the Current directory must be reflected in the stable storage, as they define the current state of the stable database and therefore must be available after system failure. In addition, for each active transaction T_i there is a private directory which stores the locations of the new versions written by T_i . When a transaction T_i is ready to commit, the Recovery Manager updates the Scratch directory to include T_i 's updates (recorded in the private directory). Then it swaps the Current and Scratch directories in an atomic action, by swapping the pointer in the Master record. This allows only one commit to be processed at a time [BERN87]. Therefore, the process of updating the Scratch directory followed by swapping the pointer in Master record must be done without interruption. If a low-level transaction wishes to commit while a high-level transaction is in its commit phase, then the low-level transaction must wait for the high-level transaction to complete. However, this violates the MLS property and thus, an RM which employs the *Shadow Page Algorithm* is insecure.

6.3. A recovery protocol for an MLS DBMS

We now consider a recovery protocol which is secure. First we need to define the interface between the RM and the CM. We assume the CM supports the following operations:

fix_new(opi , obj) : Allocate and pin an empty slot in the cache for object obj then return the slot number to the requesting operation opi . This is used to create a new object.

fix(opi , obj) : Locate and pin a slot in the cache containing obj and return the slot number to the operation opi .

write(opi , c , v) : Write the value v into cache slot c for the operation opi .

read(opi , c) : Read the contents of the object which currently occupies slot c and return the value to the operation opi .

flush(opi , obj) : Flush the object obj from the cache. Flush has no effect if the object is not in the cache.

unfix(opi , c) : Unpin the cache slot c .

¹Our definition of Atomic and ~Atomic propagation strategy differ slightly from the definitions given in [HAER83]

appendlog(op_i, v) : Append a record with value *v* to the log on behalf of operation *op_i*.

We call the actions described above *concrete actions*. Note that each concrete action keeps track of the operation *op_i* from transaction *T_i* which issues it. We call these transaction operations *abstract actions*.

A cache slot can be pinned by more than one transaction. When a cache slot *c* is pinned by a transaction, the Cache Manager cannot select *c* as a victim for replacement until all transactions have unpinned the slot.

6.3.1. Analysis of the Undo/No-Redo protocol

If the recovery manager employs update in-place and an Undo/No-Redo protocol, it must undo aborts by restoring before images which would require strict scheduler. So this class of RM is not suitable for Class 2-SS transactions. Here we consider the Undo/No-Redo protocol for use with multiversion schedulers. In this case, strictness is not required.

We can describe the recovery protocol in terms of the above concrete actions and some internal computations. We will refer to the RM procedures for Write, Read, Commit and Abort as abstract actions. When an abstract action arrives at the RM, it will trigger some internal computations, output a sequence of concrete actions to the CM and receive a sequence of acknowledgments. When a concrete action is output, the computation is blocked until acknowledgment is received. While awaiting the acknowledgment, the RM can process the next input action. If a variable is assigned the return value of a concrete action, we mean that the variable is assigned the return value contained in the acknowledgment of this concrete action.

Under our model for a transaction processing system, the acknowledgments from the Recovery Manager are part of the inputs to the scheduler. Also, the acknowledgments from the Cache Manager are part of the inputs to the Recovery Manager, and similarly for the Cache Manager and Disk Manager. Thus, we capture the notion of concurrency within a subsystem by the interleaved execution of several operations. An operation is considered complete only when its acknowledgment from the next subsystem is received.

In the following procedures, *objseq[0..N]* is an array in which *objseq[i]* contains the sequence of objects written by transaction *T_i*.

RM-Write(transaction *t_i*; item *x*; version *i*; value *v*): acknowledgment

```
objseq[i] := objseq[i] | addr(x, i);  
c := fix_new(Wi(x, i, v), addr(x, i));  
write(Wi(x, i, v), c, v);  
unfix(Wi(x, i, v), c);  
return(Ack(RM-Write(ti, x, i, v)) );
```

The first statement in RM-Write appends the address of object (version) *x[i]* to the end of the sequence *objseq[i]*, where the function *addr(x, i)* computes the address of an object given item name *x* and version number *i*. The notation *W_i(x, i, v)* appeared in the *concrete actions* (*Italic letters*) means the requesting *abstract action* is a write to version *x[i]* with value *v*.

RM-Read(transaction *t_i*; item *x*; version *j*): acknowledgment

```
c := fix(Ri(x, j, null), addr(x, i));
```

```

v := read(Ri(x, j, null), c);
unfix(Ri(x, j, v), c);
return(Ack(RM-Read(tj, x, j, v)) );

```

Note that the acknowledgment of RM-Read must contain a return value. So the notation $R_i(x, j, v)$ appeared in the concrete actions means the requesting action is a read of version $x[j]$ with return value v . Before the object is actually read, the value of v will be null. After the CM acknowledges RM's read action, the return value is stored in the variable v and is subsequently passed to the scheduler.

RM-Commit(transaction t_j): acknowledgment

```

For obj in objseq[i] do
    flush(Ci, obj);
appendlog(Ci, (tj, commit, objseq[i]));
return(Ack(RM-Commit(tj)));

```

The RM-Commit flushes objects from the cache in the order the transaction wrote them, and appends a commit record to the log. It then returns an acknowledgment to the scheduler.

RM-Abort(transaction t_j): acknowledgment

```

appendlog(Ai, (tj, abort));
return( Ack(RM-Abort(tj) ) );

```

The RM-Abort simply appends an abort record to the log and returns an acknowledgment.

A multiversion scheduler maps a Read to an appropriate version and maps a Write to a new version. It maintains a version table storing the version numbers for each data item. Since unused versions can be garbage collected, and the number of versions can be bounded by the degree of multiprogramming, we assume that the version table is stored in primary memory. Each version is tagged by the name of the transaction that created it. The transaction names are a set of totally ordered identifiers. The scheduler must use the version table to find an appropriate version for a Read. When the scheduler decides to abort a transaction T_i , it must make sure the versions created by T_i in the version table are removed.

We assume that the Recovery Manager can compute the address in the stable database of a version given its name and version number, therefore, the function $addr(x, i)$ can compute the address of the version $x[i]$ using only the item name x and the version number i . One way to achieve this is to maintain a lookup table with an entry for each useful versions. The entry includes the disk address of the element. There is a default location on disk for each data item. Only the last committed versions and active versions need to be maintained in the table. So, the table will be much smaller than the size of the database. The last committed version for each database item can be recovered from stable storage after system failure as we will see in the Restart procedure.

RM-Restart(): acknowledgment

1. Discard the contents of all cache slots;
2. Scan the log and construct the *commit_list* and *objseq* from the commit records;
3. Reconstruct the final state of the database:
 Let $CL := commit_list$;

ItemNotDone := the set of database item names;
 While *CL* \neq EMPTY and *ItemNotDone* \neq EMPTY do the following:
 let *n* := the transaction name with maximum timestamp in *CL*;
 For each *x* in *objseq*[*n*] and in *ItemNotDone*
 Set last committed version of *x* to *n*;
 CL := *CL* - {*t_n*};
 ItemNotDone := *ItemNotDone* - {*objseq*[*n*]};
 End while;
 Reconstruct the version table from the last committed version of every item;
 4. Free the space for the versions in stable storage which are not the last committed versions;
 5. Acknowledge to the scheduler the completion of Restart;

6.3.2. Correctness

We say a recovery protocol is correct if after a system failure, recovery should leave the database in the same state as the committed projection of the execution which precedes the failure. To achieve this, it is necessary that the recovery protocol satisfy the undo and redo rule and the RM outputs only serializable and recoverable schedules. This algorithm satisfies the undo rule, because the last committed version is never overwritten. It also satisfies the redo rule for the simple reason that it does not require redo. All writes are flushed to the stable database before the transaction that issued them commits. Therefore, the last committed versions are always available in the stable storage for *restart* after a system failure. In step (3) of RM-Restart, the last committed version of *x* is restored if *x* is written by a committed transaction and no other committed transactions wrote a larger version of *x*. Correctness here is based on the rule that the highest committed versions constitutes the final state of the database. Thus, when *restart* terminates, the last committed version of each data item is restored. The version table is reconstructed from the last committed versions for the scheduler as its new initial state. The *restart* is also idempotent, meaning any incomplete execution of *restart* followed by a complete execution has the same effect as one complete execution. This is true because the *restart* does not change the information in the log until it is completed. The changes to the database made by an incomplete *restart* will be recovered from the log by the last completed one.

We assume the scheduler produces serializable and recoverable schedules and never outputs two conflicting (abstract) actions to the RM at the same time. Two actions conflict whenever one modifies an element and the other observes its state. We can show that no additional synchronization is necessary for the RM executing the above protocol to ensure that no two conflicting actions are dispatched to the CM.

Lemma 3- A multiversion scheduler which never sends two conflicting (abstract) actions to the RM using the Undo/No_Redo protocol described above, outputs no conflicting (concrete) actions simultaneously to the CM and makes no conflicting accesses to shared variables, if no synchronization is enforced except that provided by the scheduler.

Proof: If the RM does not enforce any synchronization, we must make sure that there are no conflicting accesses to the shared variables in the RM and no conflicting concrete actions are issued to the CM for all possible execution traces. Note that the only shared variable in the RM is *objseq*. We examine the interaction between all possible pairs of the RM procedures. Here we assume the actions in the same transaction execute serially but the execution of actions from different transactions can be overlapped.

Case 1. The execution of RM-Write(*t_i*, *x*, *i*, *v*) overlaps that of RM-Write(*t_j*, *x*, *j*, *v*)
 for *i* \neq *j*.

Because version $x[i]$ and $x[j]$ are different objects, they cannot share a common cache slot c . Thus, the concrete actions operate on different slots and do not conflict. Moreover, access to the shared table *objseq* will not conflict, since *objseq[i]* and *objseq[j]* represent separate elements in the table.

Case 2. The execution of $\text{RM-Write}(t_i, x, i, v)$ overlaps that of $\text{RM-Read}(t_j, x, k)$ for $i \neq j$.

They do not conflict when $k \neq i$ because $x[i]$ and $x[k]$ represent different objects. Thus, each concrete action operates on a different slot. The case when $k = i$ is excluded by the fact that the scheduler never sends two conflicting abstract actions to the RM.

Case 3. The execution of $\text{RM-Write}(t_i, x, i, v)$ overlaps that of $\text{RM-Commit}(t_j)$ for $i \neq j$.

They do not conflict because object $x[i]$ is not in the object sequence *objseq[j]* written by t_j because we do not allow a version to be overwritten. The *appendlog* does not conflict with any action since each invocation of *appendlog* writes a new log record.

Case 4. The execution of $\text{RM-Write}(t_i, x, i, v)$ overlaps that of $\text{RM-Abort}(t_j)$ for $i \neq j$.

They do not conflict since *appendlog* does not conflict with any other action. Access to *objseq[i]* and *objseq[j]* do not conflict since they represent different elements.

Case 5. The execution of $\text{RM-Read}(t_i, x, k)$ overlaps that of $\text{RM-Read}(t_j, x, n)$ for $i \neq j$.

They do not conflict because there is no access to a shared variable.

Case 6. The execution of $\text{RM-Read}(t_i, x, k)$ overlaps that of $\text{RM-Commit}(t_j)$ for $i \neq j$.

They do not conflict when $k \neq j$ because object $x[k]$ is not in the object sequence *objseq[j]* written by t_j . If $k = j$ then the object being read, $x[k]$, is in the object sequence *objseq[j]* written by transaction t_j , therefore it is possible that t_j flushes object $x[k]$ which occupies the slot currently being read by another transaction t_i . However, the two accesses to the slot do not conflict.

Case 7. The execution of $\text{RM-Read}(t_i, x, k)$ overlaps that of $\text{RM-Abort}(t_j)$ for $i \neq j$.

They do not conflict because *appendlog* does not conflict with any action and there is no access to a shared variable in RM-Abort .

Case 8. The execution of $\text{RM-Commit}(t_i)$ overlaps that of $\text{RM-Commit}(t_j)$ for $i \neq j$.

They do not conflict since *objseq[i]* and *objseq[j]* are different elements in the *objseq* shared variable. The objects in *objseq[i]* and *objseq[j]* do not intersect because no versions can be overwritten, thus no two actions flush the same object.

Case 9. The execution of $\text{RM-Commit}(t_i)$ overlaps that of $\text{RM-Abort}(t_j)$ for $i \neq j$.

They do not conflict since i and j address different elements of *objseq*.

Case 10. The execution of $\text{RM-Abort}(t_i)$ overlaps that of $\text{RM-Abort}(t_j)$ for $i \neq j$.

They do not conflict since *appendlog* operations never conflict.

Therefore, we conclude that no conflicting actions are output to the CM concurrently and no conflicting accesses to a shared variable are made at the same time. Thus, output actions are dispatched immediately without delay. \square

If the recovery algorithm is correct when each abstract action is executed atomically, it will also be correct if the synchronization provided by the scheduler is the only synchronization. Therefore, Lemma 3 implies that all possible interleavings of concrete actions output by the RM are correct, and the RM never needs to delay actions.

6.3.3. Security

We now examine the security property of the undo/no-redo protocol described above.

Theorem 4- A RM following the Undo/No-Redo protocol described above is MLS.

Proof: The intuition is based on the following observations:

1. The RM never delays actions.
2. The input-output mappings for Write, Read and Abort are state independent. The mapping for a Commit operation only depends on the issuing transaction.

We need to prove the outputs of RM satisfy $\text{purge}(l, \hat{\lambda}(s_0, p)) = \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p)))$ for all possible inputs p . We prove this by an induction over the length of the input p .

Basis. $p = \phi$.

$$\text{purge}(l, \hat{\lambda}(s_0, p)) = \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p))) = \phi.$$

Induction. Assume $\text{purge}(l, \hat{\lambda}(s_0, p')) = \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p')))$ holds for $|p'| < n$. Let $p = p'la$ and $s_{p'} = \delta(s_0, p')$ be the state after input p' . We have

$$\begin{aligned} & \text{purge}(l, \hat{\lambda}(s_0, p'la)) \\ &= \text{purge}(l, \hat{\lambda}(s_0, p') \cup \lambda(s_{p'}, a)) \quad \dots \text{by def. of extended output function} \\ &= \text{purge}(l, \hat{\lambda}(s_0, p')) \cup \text{purge}(l, \lambda(s_{p'}, a)) \quad \dots \text{by purge property (2)} \end{aligned}$$

Similarly, let $s_{q'} = \delta(s_0, \text{purge}(l, p'))$ be the state following the input $\text{purge}(l, p')$. We have

$$\begin{aligned} & \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p'la))) \\ &= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p'))) \cup \text{purge}(l, \lambda(s_{q'}, \text{purge}(l, a))) \end{aligned}$$

Case 1. $\text{level}(a) \leq l$.

Because the RM never delays actions, an output action of the RM is labeled with the classification level of its corresponding input. We have:

$$\text{purge}(l, \lambda(s_{p'}, a)) = \lambda(s_{p'}, a) \text{ and } \text{purge}(l, \lambda(s_{q'}, \text{purge}(l, a))) = \lambda(s_{q'}, a). \quad \dots(1)$$

Case 1.1. input a is Commit or the acknowledgment of a flush action for some transaction T_i .

The output $\lambda(s_p, a)$ depends on the state component $s_p.objseq[i]$, which is the value of the sequence $objseq[i]$ in state s_p . Similarly, $\lambda(s_q, a)$ depends on $s_q.objseq[i]$. Observe that in the procedures of Undo/No-Redo protocol, only transaction T_i can modify $objseq[i]$. Since $level(T_i) \leq l$, the actions of T_i are not purgeable and therefore the sequence of T_i 's actions in schedules p' and $purge(l, p')$ are identical, we have $s_p.objseq[i] = s_q.objseq[i]$ by the induction hypothesis. If a is RM-Commit(T_i) then output $\lambda(s_p, a) = flush(C_i, obj) = \lambda(s_q, a)$, where obj is the first element in $objseq[i]$. If a is the j th $Ack(flush(C_i, obj))$ following RM-Commit(T_i) then output $\lambda(s_p, a) = flush(T_i, obj[j]) = \lambda(s_q, a)$, where $obj[j]$ is the j th element following obj in $objseq[i]$. If j equals the number of elements in $objseq[i]$, then output $\lambda(s_p, a) = appendlog(C_i, (T_i, commit, objseq[i])) = \lambda(s_q, a)$.

Case 1.2. a is an input action other than those in Case 1.1.

The output actions are determined by the input-output mapping of RM which are fixed in this case. We list all possible actions for input a and its corresponding output $\lambda(s, a)$ below:

a	$\lambda(s, a)$
RM-Write(t_i, x, i, v)	$fix_new(W_i(x, i, v), addr(x, i))$
$Ack(fix_new(W_i(x, i, v), obj), c)$	$write(W_i(x, i, v), c, v)$
$Ack(write(W_i(x, i, v), c, v))$	$unfix(W_i(x, i, v), c)$
$Ack(unfix(W_i(x, i, v), c))$	$Ack(RM-Write(t_i, x, i, v))$
RM-Read(t_i, x, j)	$fix(R_i(x, j, null), addr(x, j))$
$Ack(fix(R_i(x, j, null), obj), c)$	$read(R_i(x, j, null), c)$
$Ack(read(R_i(x, j, v), c))$	$unfix(R_i(x, j, v), c)$
$Ack(unfix(R_i(x, j, v), c))$	$Ack(RM-Read(t_i, x, j, v))$
RM-Abort(t_i)	$appendlog(A_i, (t_i, abort))$
$Ack(appendlog(A_i, v))$	$Ack(RM-Abort(t_i))$
$Ack(appendlog(C_i, v))$	$Ack(RM-Commit(t_i))1$

Note that the function $addr(x, i)$ only depends on the item name x and version number i which are supplied by the input arguments. Therefore, function $addr$ which computes the address of a version is state independent. Because the RM never delays actions, we have:

$$\lambda(s_p, a) = \lambda(s_q, a) \text{ for all } s_p \text{ and } s_q. \quad \dots(2)$$

By the induction hypothesis $purge(l, \hat{\lambda}(s_0, p')) = purge(l, \hat{\lambda}(s_0, purge(l, p')))$.

We conclude

¹This is the final acknowledgement for a commit action.

$$\begin{aligned}
& \text{purge}(l, \hat{\lambda}(s_0, p' | a)) \\
&= \text{purge}(l, \hat{\lambda}(s_0, p') \cup \lambda(s_p', a)) && \dots \text{by def. of extended output function} \\
&= \text{purge}(l, \hat{\lambda}(s_0, p')) \cup \text{purge}(l, \lambda(s_p', a)) && \dots \text{by purge property (2)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, p')) \cup \lambda(s_p', a) && \dots \text{by (1)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p'))) \cup \lambda(s_q', a) && \dots \text{by induction hypothesis and (2)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p'))) \cup \text{purge}(l, \lambda(s_q', \text{purge}(l, a))) && \dots \text{by (1)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p')) \cup \lambda(s_q', \text{purge}(l, a))) && \dots \text{by purge property (2)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p' | a))) && \dots \text{by def. of extended output function}
\end{aligned}$$

Case 2. $\text{level}(a) \neq l$.

We have $\text{purge}(l, \lambda(s_p', a)) = \tau_a = \text{purge}(l, \lambda(s_q', \text{purge}(l, a)))$ because the output in response to a is labeled with the classification $\text{level}(a)$, it follows that

$$\begin{aligned}
& \text{purge}(l, \hat{\lambda}(s_0, p' | a)) \\
&= \text{purge}(l, \hat{\lambda}(s_0, p') \cup \lambda(s_p', a)) && \dots \text{by def. of extended output function} \\
&= \text{purge}(l, \hat{\lambda}(s_0, p')) \cup \text{purge}(l, \lambda(s_p', a)) && \dots \text{by purge property (2)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, p')) \cup \tau_a && \dots \text{by } \text{purge}(l, \lambda(s_p', a)) = \tau_a \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p'))) \cup \tau_a && \dots \text{by induction hypothesis} \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p'))) \cup \text{purge}(l, \lambda(s_q', \text{purge}(l, a))) && \dots \text{by } \text{purge}(l, \lambda(s_q', \text{purge}(l, a)) = \tau_a \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p')) \cup \lambda(s_q', \text{purge}(l, a))) && \dots \text{by purge property (2)} \\
&= \text{purge}(l, \hat{\lambda}(s_0, \text{purge}(l, p' | a))). && \dots \text{by def. of extended output function}
\end{aligned}$$

We have established Theorem 4. □

7. CONCLUSION

In this paper, we introduce a model for a Transaction Processing System (TPS). This model captures the real time behavior of the system in the sense that each input is labeled with its arrival time and each output is marked with its departure time. Based on this model, we introduce a security (MLS) property based on *noninterference*. We show that the MLS property is composable for our model. We also show how to construct a TPS using only parallel composition on feedback. This allows us to decompose a large system into smaller subsystems which simplifies the analysis of security for each subsystem. Based on the notion of MLS composability we have shown that a TPS is MLS if the scheduler, recovery manager, cache manager and disk manager are all MLS.

We have analyzed the security properties of various recovery protocols. Most existing recovery protocols rely on the scheduler to output *strict* schedules. We have shown that no strict scheduler is secure with respect to Class 2-SS transactions. An RM which employs Update In-Place and undoes aborts by replacing before images, requires a scheduler which outputs strict schedules. This implies the Undo/Redo and Undo/No_Redo protocols as described in [BERN87] when used with single version schedulers are not secure for Class 2-SS

transactions. The No-Undo/No-Redo protocol (*Shadow Page Algorithm*) allows only one transaction to commit at a time, and so we have argued that it is insecure.

We have described an Undo/No_Redo protocol suitable for multiversion schedulers. We have shown this protocol is both correct and secure.

We assume that the location of versions of data elements can be computed or maintained in memory. This simplifies our modeling. In the future we would like to investigate other solutions to this problem.

It is interesting to note that the actions taken by our recovery protocol depend solely on the requesting transaction. For this reason, we believe an untrusted implementation may be feasible. This will be the subject of future work.

8. BIBLIOGRAPHY

- [BELL76] D.E. Bell, and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretations," Technical Report MTR-2997, Mitre Corp., March 1976.
- [BERN87] P.A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
- [COST91] Oliver Costich, "Transaction Processing Using an Untrusted Scheduler in a Multilevel Database with Replicated Architecture," *IFIP WG 11.3, Fifth Working Conference On Database Security*, Shepherdstown, WV, Nov. 1991.
- [DOD85] Department of Defense Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.
- [DOWN89] A.R. Downing, I.B. Greenberg and T.F. Lunt, "Issues in Distributed Database Security," *Proceedings of the Fifth Annual Computer Security Applications Conference*, Tucson, AZ, December 1989, pp. 196-203.
- [EFFE84] W. Effelsberg and T. Haeder, "Principals of Database Buffer Management.", *ACM Transaction on Database Systems* 9(4):560-595, Dec, 1984.
- [GOGU82] J.A. Goguen and J. Meseguer, "Security Policy and Security Models," *Proceedings of the IEEE Symposium on Security and Privacy*, 1982, pp. 11-20.
- [GOGU84] J.A. Goguen and J. Meseguer, "Unwinding and Inference Control," *Proceedings of the IEEE Symposium on Security and Privacy*, 1984, pp. 75-86.
- [GREE91] Ira Greenberg, "Distributed Database Security," *Technical Report, SRI Project 8772*, April 1991.
- [HAIG87a] J.T. Haigh and W.D. Young, "Extending the Noninterference Version of MLS for SAT," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987, pp. 141-150.

- [HAIG88] J.T. Haigh, P.D. Stachour, P.A. Dwyer, E. Onuegbu, M.B. Thuraisingham, "Secure Distributed Data Views (LDV): Implementation Specification for a Database Management System," A005: Interim Report, Honeywell, May 1988.
- [HAER83] T. Haerder and A. Reuter, A. "Principals of Transaction-Oriented Database Recovery" *ACM Computing Surveys* 15(4): 287-317, Dec., 1983.
- [JAJ090] Sushil Jajodia and Boris Kogan, "Transaction Processing in Multilevel-Secure Databases Using Replicated Architecture," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1990, pp. 360-368.
- [KANG92] I.E. Kang and T.F. Keefe, "Recovery Management for Multilevel Secure Database Systems", Technical Report TR-92-103, Dept. of Electrical and Computer Engineering, Pennsylvania State University, March 1992.
- [KARG91] P.A. Karger and J.C. Wray, "Storage Channels in Disk Arm Optimization", *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1991, pp. 52-61.
- [KEEF92] T.F. Keefe and W.T. Tsai, "Database Concurrency Control in Multilevel Secure Database Management Systems," to appear in *IEEE Transaction on Knowledge and Data Engineering*.
- [KEEF90a] T.F. Keefe, and W.T. Tsai, "Multiversion Concurrency Control for Multilevel Secure Database Systems," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1990, pp. 369-383.
- [KEEF90b] T.F. Keefe, "Multilevel Secure Database Management Systems," Ph.D. Dissertation, University of Minnesota, 1990.
- [LUNT90] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman and W.R. Shockley, "The Sea View Security Model," *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, June 1990, pp. 593-607.
- [MAIM90] William T. Maimone and Ira B. Greenberg, "Single-Level Multiversion Schedulers for Multilevel Secure Database Systems," to appear in *Proceedings of the Sixth Annual Computer Security Applications Conference*, Tucson, AZ, December 1990.
- [MILL90] Jonathan K. Millen, "Hookup Security for Synchronous Machines," *Proceedings of the Computer Security Foundations Workshop III*, Franconia, NH, June 1990, pp. 84-90.
- [MCCU90] Daryl McCullough, "A Hookup Theorem for Multilevel Security," *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, June 1990, pp. 563-568.
- [PAPA86] C. Papadimitriou, *The Theory of Database Concurrency Control*, Computer Science Press, Rockville, MD, 1986.
- [SHOC89] William R. Shockley, Daniel Warren, Terry C. Cheung and David R. Schell, "Secure Distributed Data Views System Specification," RADC-TR-89-313,

Vol. V, Final Technical Report, Computer Science Lab., SRI International, Menlo Park, CA., December 1989.

- [OBRI90] Richard C. O'Brien, J.T. Haigh, D.J. Thomsen, "Trusted Database consistency Policy," RADC-TR-90-387, Final Technical Report, Secure Computing Technology Corporation, December 1990.
- [VETT89] L. Vetter, G. Smith and T.F. Lunt, "TCB Subsets: The Next Step," *Proceedings of the Fifth Annual Computer Security Applications Conference*, Tucson, AZ, December 1989, pp. 216-221.
- [WITT90] J. Todd Wittbold and Dale M. Johnson, "Information Flow in Nondeterministic Systems," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1990, pp. 144-161.

Maintaining Multilevel Transaction Atomicity in MLS Database Systems with Kernelized Architecture

Oliver Costich*
Sushil Jajodia

Center for Secure Information Systems, George Mason University, 4400
University Drive, Fairfax, Virginia 22030, USA

Abstract

In most models of trusted database systems, transactions are considered to be single-level subjects. As a consequence, users are denied the ability to execute some transactions which can be run on conventional (untrusted) database systems, namely those that perform functions that become inherently multilevel in the MLS environment. This paper introduces a notion of multilevel transaction and proceeds to an algorithm for their concurrent execution. The algorithm is proven to be correct in the sense that resulting schedules for executing the multilevel transactions is one-copy serializable.

1. INTRODUCTION

Most approaches to transaction processing for trusted database systems (TDBS) do something like the following. There is a set of *data items*, labeled with security classes from a lattice of security classes (or levels), which serve as the *objects* of the system. There is another set, of *transactions*, also labeled with security classes, in the role of the *subjects* of the system. A mandatory access control policy

*The work of this author was supported in part by the Naval Research Laboratory under Contract N0001489-C-2389

is adopted that enforces the Simple Security and \star -property of Bell and LaPadula [1]; namely subjects may write to objects only if the label of the subject is dominated by that of the object, and subjects may read from objects only if the label of the subject dominates that of the object. (Frequently the write condition is restricted to permit a subject to write to an object only if the labels are the same.) From the point of view of the security world, then, subjects and objects are the atomic units of interest. The approach described above enforces this point of view on the database system (and its users) as well.

On the other hand, in the database world, a different view of what constitutes atomicity prevails. Data items remain the elements of interest from the users' point of view. However, **DBS** users see two types of entities that operate on the data items. First there are *read* and *write operations* that are applied directly to data items. They also construct *transactions* as sequences of these operations that the users' expect to be executed atomically on the database. That is, a transaction is either executed completely and the resulting changes to the values of the data items made permanent, or the transaction has no effect at all. In addition, these transactions are independent in that there is no communication among transactions except through their effect on the values of data items. There is no external communication among transactions.

At first glance, the views of the security world and the database world seem in agreement. However, a conflict between them does exist. Implicit in the security view is that transactions have a unique security level. That is, subjects are single-level. From the database users' view, operations on data items are single-level, but requiring entire transactions to be so may be inadequate for many transactions that they may want to use. Examples may help.

A satellite uses sensors to collect sensitive information in its scanning range. That data, together with the position of the satellite, is used by an analytical process. The position data has security level **U** (unclassified) while the other data and the result of the analysis has security level **S** (secret). The security world would like to split this into two transactions; a **U** transaction that records the position data and an **S** transaction that then reads the position data, retrieves the other data, performs the analysis, and finally writes the result to the database. To do this, the user would have to log-on to the system at level **U** and submit the first transaction, then log-out and log-in at level **S**, where the second transaction would be submitted. The database world would like to do this using only a single transaction since it must be done as a single atomic action to insure that the result be correct. The two transactions approach embodies the following pitfall. Since the two transaction cannot communicate except through their action on data items, there is no assurance that the second transaction will read the position data submitted by the first. Incorrect data could be read by the second transaction in two ways. The update to the position data may not have been made by the time the second transaction reads the position data item and so the old

position would be used in the analysis. This can be overcome by the user waiting for a commitment message from the system before logging out from the U level. But there is another way that incorrect data can be read by the second transaction. Namely, newer position data is written by a different user's transaction before the correct data is read by the second transaction and again the analysis is incorrect. This cannot be corrected without the two transactions communicating in some external way.

The situation can get more complex, as shown in this second example. Suppose a company records the hours worked by each employee and computes the employees' salaries for the pay period. The hours data has security level U (unclassified) while the hourly rate data and the gross salary data have security level S (secret). The transaction to be performed first updates the hours worked from the time card, and then retrieves the hourly rate and computes the salary. Finally, the hours worked data item is reset to zero. The security world technique would require three transactions. The first updates hours worked. The second retrieves the rate and computes the salary. The third resets the hours worked to zero. Three distinct log-ins are required, and the first and second have the same problem as our previous example. Beyond that if the third were completed before the second transaction retrieved the hours data, the salary would be calculated incorrectly.

In the conventional (not trusted) database world, these problems would not exist, because the user could submit these combined actions as a single transaction. Ordinary concurrency control mechanisms (which enforce *serializability* of collections of transactions) would insure that correct values were read and written in the proper sequence.

It appears that some notion of multilevel transaction is required to resolve this dilemma. Previous work in this direction for a limited class of multilevel transactions and for replicated architecture multilevel database systems appears in [4]. Here we intend to extend the idea of multilevel transaction to a significantly larger class of multilevel transactions, one that encompasses virtually every situation that we can construct. We will formally define notions of multilevel transaction and the correctness of their execution (serializability) in centralized multilevel database systems with kernelized architecture. A scheduling algorithm will be presented and shown to be correct.

2. THE SECURITY MODEL

The architecture for the systems under consideration is based on one that is frequently proposed [7,8]. The security features are enforced by a security kernel,

the trusted computing base of the trusted operating system, together with whatever additional trusted processes are necessary in the database application to enforce the overall system's security policy. The idea is to minimize the trusted processes required to do this.

The security policy for our system will be a variant of the mandatory access control policy of Bell and LaPadula [1]. There is a set **D** of data items of the database system that serve as the objects of the multilevel system. The subjects of the system, denoted **Sub**, are quite similar to the single-level transactions used in earlier work [3,6,7,8,9,10]. However our transactions will be more complex and will be formed by interleaving the subjects of the database system.

More formally, we limit operations on the data items. Only Reads, denoted $r[x]$, and Writes, denoted $w[x]$ together with Aborts, denoted **a**, or commits, denoted **c** are considered. A subject of **Sub** is a sequence of Reads and Writes ending with either an Abort or a Commit (but not both). There is a lattice $(SC, <)$ of security labels and a function L , mapping subjects and objects into security classes, i.e., $L:D \cup \text{Sub} \rightarrow SC$. The security policy has two conditions:

(1) (Simple Security Property) If $T \in \text{Sub}$ and $r[x] \in T$, then $L(x) \leq L(T)$.

(2) (Restricted \star -Property) If $T \in \text{Sub}$ and $w[x] \in T$, then $L(x) = L(T)$.

That is, subjects can read from dominated security levels, but only write at their own security level. These are basically the mandatory access control policies of [1], slightly modified.

3. THE TRANSACTION MODEL

To define multilevel transaction, we need some preliminaries. A data item x can take on values from its domain, $\text{dom}(x)$. A state of the database is determined by assigning each x in **D** a value from its domain, i.e., the states are functions $f:D \rightarrow \cup\{\text{dom}(x) | x \in D \text{ and } f(x) \in \text{dom}(x)\}$. **V** will denote the set of all such functions. Further, let $D_{\leq l} = \{x \in D | L(x) \leq l \text{ and } l \in SC\}$ and let $V_{\leq l}$ be obtained by restricting each $f \in V$ to $D_{\leq l}$ (denoted $f_{\leq l}$). Notice that any action on the database defines a mapping of **V** to **V**. In particular, if **T** is a transaction and $f \in V$, then $(T(f))(x)$ is the value of x resulting from executing **T** on the database starting with the values of the data items specified by **f**. $T_{\leq l}$ will denote **T** restricted to $V_{\leq l}$. Alternatively, $T_{\leq l}$ is the transaction obtained by discarding the operations not dominated by l (and keeping the implied order).

Definition A *multilevel transaction* T_i is a sequence^{**}, ordered by \leq , of ordered pairs of the form (o_i, l) where o_i is one of $a_i, c_i, r_i[x], w_i[x]$ for some $x \in D$ and $l \in SC$ that satisfies the following conditions:

- (1) Either $\{(c_i, l) | l \in SC\} \subseteq T_i$ or $\{(a_i, l) | l \in SC\} \subseteq T_i$, but not both.
- (2) Let e_i be either a_i or c_i . Then for each $l \in SC$, $(o_i, l) \leq (e_i, l)$.
- (3) For $f \in V$, we have $T_{i|_{\leq l}}(f_{\leq l}) = (T_i(f))_{\leq l}$ for each $l \in SC$.

The first condition requires a multilevel transaction to commit at each security level or abort at each security level. Security considerations alone would only require that no commits occur at security levels higher than one at which an abort had occurred, else lower level subtransactions would have to be rolled back to insure atomicity. Imposing this condition guarantees that this possibility is avoided. We should point out that we are only accounting for aborts due to concurrency control considerations and not for those due to violations of integrity constraints, such as range constraints. Aborts for other reasons are problematic regardless of the concurrency control technique employed. The second condition forbids further operations to be done at a given security level after the commit or abort at that level.

The third condition is more difficult to explain. Notice that for a multilevel operation (o_i, l) , o_i can only operate on a data item whose security level is dominated by l in SC . This means that operations in $T_{i|_{\leq l}}$ are only applied to data items at the level of l or below, and that $T_{i|_{\leq l}}(f_{\leq l})$ is the result of applying these operations to those data items. $(T_i(f))_{\leq l}$ is the result of executing T on the entire database and then looking only at the result on the data items with security level l or below. The equality in the condition says that the values of data items at level l and below which result from executing T depend only on the values of those data items when T was initiated, and not on the values of any higher level data items. Said differently, no information about the value of higher level data items can flow to lower level data items by virtue of running the transaction.

Consider our earlier examples in light of this definition. The first example becomes $(w[x], U)(c, U)(r[x], S)(r[y], S)(w[z], S)(c, S)$ where $L(x)=U$, and the other data items have security level S . This clearly satisfies the conditions. Condition (3) is satisfied since the transaction never writes to a lower level data item after

^{**}We could use a definition of transaction based on partial orders, as in [2]. However the results are actually no more general but the definitions, the algorithm, and the proofs are more complicated. We will use sequences rather than partial orders throughout this paper since it simplifies the explication with no loss of generality.

accessing a higher level one. Transactions of this form are treated in [4] for a different architecture.

Our second example becomes (omitting the commit operations for simplicity), $(w[x], U)(r[y], S)(r[y], S)(w[z], S)(w[x], U)$. Unlike the prior example, this transaction writes a lower level data item after reading a higher level one. But since the second time x is written the value is always zero, the last condition is satisfied, and we have a legitimate multilevel transaction.

Whether the third condition is satisfied is not easily determined by the **TDBMS** itself. One way to resolve this difficulty is to limit transactions to those that satisfy a more restrictive, but more easily detectable form (as in [4], for example).

Another solution, which we believe is more likely, is to restrict multilevel transactions to predefined transactions that can be determined ahead of time and verified to satisfy this condition. Ad hoc user defined transactions would not be allowed because of the risk of violating the condition. Under this approach, the data items on which a transaction would operate would be known at the time the transaction is submitted to the system. The algorithm presented here relies on this assumption.

Operations of several transactions can be commingled so that concurrency of execution can be extended to sets of transactions, as reflected in the following.

Definition A complete multilevel history H over a set of multilevel transactions $T = \{T_1, T_2, \dots, T_n\}$ is a sequence with ordering relation $<_H$ where

- (1) There is a multilevel T_0 that precedes all other transactions. T_0 has operations $\{(w_0[x], I) | x \in D\}$.
- (2) $H \supseteq T_0 \cup T_1 \cup T_2 \cup \dots \cup T_n$
- (3) $<_H \supseteq <_0 \cup <_1 \cup <_2 \cup \dots \cup <_n$

The first condition provides initial values of the data items, so a Read operation always succeeds some Write operation on the desired data item [11]. The second requires the history to contain precisely the operations of the original transactions. The third condition provides that the ordering of operations within the history is consistent with that of each transaction.

Notice that our notion of multilevel transaction does not limit the number of Read or Write operations on a given data item within a transaction, or even within a given level of a transaction, in contrast to the usual practice in concurrency control theory [2,11]. Since it does not, a transaction may read or write the same data item several times. We will denote the n^{th} Write operation on x by T_i by

$w_{i,n}[x]$ when necessary to avoid confusion. This multiple write capability has led us to choose a multiversion approach to concurrency control. Multiversion systems create a new version of a data item each time it is written and maintain the old versions, which does not impose significant additional burden on the DBMS, since these versions must be maintained for purposes of recovery in any case.

The preceding definition of history represents the view of the user, to whom versions of data items are transparent. The users' view represents the logical order of the execution of operations as seen by the users. Histories of this type will be called *one-copy histories* when it is necessary to distinguish them from histories that represent the system's view of a transaction and that deal with multiple versions of data items. The representation of the system's view requires a different definition of history.

When a set of transactions is executed by a multiversion DBMS, an operation in a transaction must be translated into the equivalent operation on some version of the data item. A *translation function* h performs the mapping. For a Read, h determines the version of x to be read, i.e., $h(r_i[x], l) = (r_i[x_{i,n}], l)$ where $x_{i,n}$ is the n^{th} version of x written by T_i . For a Write, h , determines what version of x will be created, i.e., $h(w_i[x], l) = (w_i[x_{i,n}], l)$ if $w[x_i]$ is the n^{th} Write operation on x in T_i .

The concept of a *multiversion data history* is needed to represent the actions of the translated transactions on the multiversion data. Recall that T_0 is an initializing transaction that writes initial values into every data item in D .

Definition A *multiversion data history* H over a set of multilevel transactions $T = \{T_1, T_2, \dots, T_n\}$ is a linear order with ordering relation $<_H$ such that

- (1) $H \supseteq h(T_0) \cup h(T_1) \cup h(T_2) \cup \dots \cup h(T_n)$
- (2) If $(p_i, l), (q_i, m) \in T_i$ with $(p_i, l) <_i (q_i, m)$ then $h(p_i, l) <_H h(q_i, m)$
- (3) For all $l \in SC$, all $i > 0$, $(w_0[x_{0,1}], l) <_H (r_{i,1}[x], l)$ for all $x \in D$.

A multiversion data history represents the order in which operations are executed by the data manager of the TDBMS on the data items stored in the TDBMS.

The first condition says that the history contains the translations of the original transactions. The second condition insures that the order of operations within transactions is preserved. The third condition provides that at each security level, T_0 initializes each data item before it is read by any of the original transactions.

Histories, one-copy or multiversion, **are complete** if they contain no operations from aborted transactions. Since we are primarily concerned with these kinds of histories, we will refer to them simply as histories. (As it turns out, our algorithm

prevents aborts, so the notions are coincident in any case.) We now turn to defining a notion of correctness for execution of multilevel transactions.

A history, one-copy or multiversion, is *serial* if for every pair of transactions T_i and T_j that appear in H , either all of the operations of T_i precede those of T_j or vice versa. In one-copy histories, correctness is defined as being equivalent to a serial history, where equivalence is, in turn, defined in terms of reads-from relationships and final writes in the usual way [2]. It is well known in the theory of database concurrency control that the parallel notion of equivalence is insufficient for multiversion transactions [2] because Read operations may now read from different versions of a data item, and a transaction may read from the correct transaction but choose the wrong version. We will now make these ideas more formal.

Definition In one-copy histories, we say $(r_j[x], m)$ *reads- x -from* $(w_i[x], l)$ if $(w_i[x], l) <_H (r_j[x], l)$ and there is no $(w_k[x], l)$ for which $(w_i[x], l) <_H (w_k[x], l) <_H (r_j[x], l)$. Notice that $m \geq l$ and that there is no requirement that i, j , and k be distinct. If $i \neq j$, we say T_j *reads- x -from* T_i , and call it a *transaction reads-from*. If $i = j$, we call it a *reflexive reads-from*. We can extend these notions to multiversion histories by considering different versions of a data item x as distinct data items, as usual.

Definition Two histories, one-copy or multiversion are, *equivalent* if they have the same transaction and reflexive read-from relationships. In the one-copy case we also require that they have the same final writes (which we will not define as we will not use equivalence for this case).

As previously mentioned, it is inadequate to require that a multiversion history be equivalent a serial multiversion history for a history to represent a correct execution of the given transactions. Something more is required. We must require that, in addition to being equivalent to a serial history, that it be equivalent to a special class of serial history.

Definition A multiversion history H is *one-copy serial* if it is serial and satisfies

- (1) If T_j *reads- x -from* T_i is a transactions reads-from then the first Read operation in T_j that reads any version of x , reads it from the version of x written by the last Write operation of T_i (note that this is well-defined since H is serial.)
- (2) If $(r_i[x_{i,n}], m)$ *reads- x -from* $(w_i[x_{i,p}], l)$ is a reflexive reads-from, then $p = n$. That is, each Read of x in T_i *reads- x -from* the version produced by the immediately preceding Write of x in T_i .

It is intuitively quite clear that one-copy serial histories are correct since they look just like the corresponding one-copy history except that the data items have versions and the Read operations are "correctly" matched to the right Write operations.

Definition A multiversion history is *one-copy serializable (1SR)* if it is equivalent to a one-copy serial multiversion history.

To show that this is an adequate criterion for correctness for (multilevel) multiversion histories, we state the following theorem without proof. A proof can be easily constructed from [2, Theorem 5.3]. Only minor changes are required to account for reflexive reads-froms.

Theorem Let H be a multiversion history over a set of multilevel transactions $T = \{T_1, T_2, \dots, T_n\}$. Then H is equivalent to a serial one-copy history over T if and only if H is 1SR.

4. THE INFORMAL PRESENTATION OF THE ALGORITHM

What must the concurrency control process in our multilevel database system do? First, it must produce a 1SR multiversion history. In addition, the way in which transactions and their operations are scheduled cannot result in the flow of high security level information to lower security level subjects. In particular, no lower security level transaction can be allowed to roll back because of the execution of a higher security level part of a transaction. We want to accomplish this with a minimum of trusted processes.

Definition Given a multilevel transaction T_i and $l \in SC$, the *l-projection* of $T_{i|l}$ of T_i is $\{(o_i, l) | (o_i, l) \in T_i\}$ with the linear ordering inherited from T_i . We refer to these *l-projections* generally as *subtransactions*.

Definition The *write set* of $T_{i|l}$ is $WS(T_{i|l}) = \{x \in D | (w[x], l) \in T_i\}$ and its *read set* is $RS(T_{i|l}) = \{x \in D | (r[x], l) \in T_i\}$. Similarly, $WS(T_{i|l}) = \{x \in D | (w[x], m) \in T_i \text{ and } m \leq l\}$ and $RS(T_{i|l}) = \{x \in D | (r[x], m) \in T_i \text{ and } m \leq l\}$.

Notice that $T_{i|l}$ is a subject of the trusted system as we have defined them, and that $T_{i|l}$ also can be viewed as a single-level transaction with security level l . Every multilevel transaction naturally gives rise to a set of subtransactions. Notice that the definition of multilevel transaction guarantees that values written at one security level cannot depend on values read at higher security levels, even if the Read precedes the Write. This means that every multilevel transaction is

equivalent to one that executes the subtransactions in an order consistent with the security lattice ordering (from low to high).

Our multiversion **TDBMS** will also have an untrusted strict multiversion timestamp order scheduler [2], P_l , for each $l \in SC$. These schedulers will be called *local schedulers* and will be used to schedule the subtransactions for their level, just as if they were single level transactions. Subtransactions, therefore, may commit (and, in theory, abort) and we refer to such as *local commits* (or *aborts*). If a subtransaction has begun execution but not locally committed, we say it is *active*.

There is also a *global scheduler* Q , which will manage subtransactions across security levels (between the local schedulers). Q will be largely untrusted, though a few trusted processes will reside there. Q will assign timestamps to transactions, compare read sets and write sets as necessary, and distribute subtransactions to the appropriate local schedulers.

Informally, the algorithm works as follows. When a multilevel transaction is received, a timestamp is assigned and Read and Write operations on each data item are indexed. I.e., the first Write of x is indexed by 1, the next is indexed by 2, and so on. Read operations receive the same index as the last preceding Write of the same data item, or 0 if there is none. The indices will allow reflexive reads-froms to find the correct version and also indicate which Read operations are involved in transaction reads-froms.

The multilevel transaction is then parsed into its subtransactions, which are distributed to the corresponding schedulers. The algorithm will execute the multilevel transaction by correctly executing the subtransactions and controlling the interleaving of subtransaction among the various transactions. The algorithm must simultaneously insure that reads-froms are executed so as to generate a **ISR** history and yet not allow information to flow from high security levels to lower ones because of concurrency control mechanisms. (In this paper, we do not address covert channels that may arise for other reasons.)

We see two ways in which transaction processing might allow high level data to be transmitted to lower security levels. First, since multilevel transactions can have Write operations that execute after higher level data items have been read by the same transaction, one must be sure that any values written after reading higher level data items do not depend on the values read at the higher levels. This is precisely what is insured by the third condition of our definition of a multilevel transaction. Second, execution of transactions must be scheduled so that no rollback of lower level or noncomparable level subtransactions can result from the scheduling mechanisms for those at higher security levels. In particular, the concurrency control algorithm cannot allow a subtransaction to abort after another subtransaction of the same multilevel transaction has been executed at

a lower level. Allowing this would require that the subtransactions at lower levels or noncomparable levels be rolled back (to satisfy the first condition of the definition of multilevel transaction), creating a covert channel.

The problem is avoided by the following technique. First, for a given multilevel transaction, subtransactions are executed in the order determined by the security lattice. That is, the subtransaction at a given security level cannot begin to execute its operations until all of the subtransactions at dominated security levels have locally committed. This guarantees that a reflexive read-from will always be able to find the correct version of the data item to be read. But it is not sufficient to insure correct schedules that avoid aborts.

Multiversion timestamp order schedulers require that each Write operation create a new version of the data item, and each Read operation will read the last version written by a committed transaction with an earlier timestamp (or from the last version written by the same transaction in the case of reflexive reads-froms). Aborts arise in such schedulers when a Write operation occurs after a Read operation on the same data item and the timestamp of the Read is later than that of the Write. That is, the Write operation has arrived too late to preserve the timestamp ordering. In such instances, the transaction requesting the Write is aborted because executing it would invalidate a Read operation that had already been performed. In our system, we cannot allow a subtransaction to locally abort if another subtransaction of the same multilevel transaction locally commits. Because the security classes form a lattice, and a transaction may have subtransactions at noncomparable security levels, to prevent covert channels we must insure that transactions never locally abort.

In other words, we must guarantee that if a subtransaction is going to write a data item, then no subtransaction with a later timestamp will ever want to read it. We must be sure that Read operations only occur after all subtransactions with later timestamps and that Write the same data item have been locally committed. Our security policy implies that it is sufficient that the local commitment criterion hold for subtransactions at security levels dominated by the level of the Read operation. The algorithm forces subtransactions to wait to start until its read set has a null intersection with the write sets of all subtractions that are active (not locally committed) at the same or lower security levels and have earlier timestamps. Notice that there is no reason to ever delay the execution of a lower level subtransaction because of a higher level subtransaction, since any reflexive reads-froms can always locate the correct version of the required data item.

Finally, though these are really implementation details, we mention how one might start a multilevel transaction, though other scenarios are possible. The user would submit the transaction to Q by logging on the system at the least upper bound of the security classes of the operations of the transaction. If there is no operation of the transaction at the level of the greatest lower bound of the

transaction's operations, then Q creates an artificial one, thereby reducing the amount of trusted code if the lowest levels of the transaction's operations are noncomparable, since then the indication that it is all right to start the transaction would be transmitted across noncomparable levels. The processing could then proceed as described above.

5. SPECIFICATION OF THE ALGORITHM

We need a few additional definitions before specifying the algorithm. We use $ts(T_i)$ to denote the timestamp assigned to the multilevel transaction T_i . Similar notation is used for the timestamps of subtransactions and operations, which inherit them from their parent transactions. We denote by $\text{lub}(i)$ the least upper bound of the security classes of the subtransactions of T_i .

Definition If $T_{i|l}$ is a subtransaction, the *conflict set* of $T_{i|l}$, $CS(T_{i|l})$, is $\cup\{WS(T_{j|l}) | T_{j|l} \text{ is active}\}$.

$CS(T_{i|l})$ is precisely the set of Write operations that have the potential to invalidate a Read operation of $T_{i|l}$, because subtransactions can only read data items at or below their own security level.

The algorithm processes transactions as follows.

At the global scheduler Q :

- I.1 The initializing transaction T_0 is received and assigned a timestamp.
- I.2 As a transaction T_i begins to be received, it is assigned a timestamp and Read and Write indices are assigned.
 - a. $(w_i[x], l)$ receives an index of 1 larger than the last preceding Write operation of T_i on x unless there are none, whence it receives an index of 1.
 - b. $(r_i[x], l)$ receives an index equal to that of the last preceding Write operation of T_i on x unless there are none, whence it receives an index of 0.
- I.3 After an operation (o_i, l) is indexed, it is sent to a single-level subprocess Q_l of Q .
- I.4 The Q_l construct the subtransactions $T_{i|l}$ for all relevant l , and form each $WS(T_{i|l})$ and $RS(T_{i|l})$.

- I.5 The Q_i distribute the $T_{i=1}$, their read and write sets, and their index information and timestamps to the corresponding local scheduler P_i .
- I.6 Commit(T_i) messages are received from $P_{lub(i)}$, and Q globally commits the multilevel transaction.

At each local scheduler P_i :

- II.1 P_i receives subtransaction packages for the $T_{i=1}$ from Q_i and determines $RS(T_{i=1}) \cap CS(T_{i=1})$.
 - a. If $RS(T_{i=1}) \cap CS(T_{i=1}) = \emptyset$, then P_i initiates execution of $T_{i=1}$, provided that $T_{i=m}$ has been locally committed for all $m < i$ in SC.
 - b. If $RS(T_{i=1}) \cap CS(T_{i=1}) \neq \emptyset$, then P_i queues $T_{i=1}$ for later execution.
- II.2 As P_i executes $T_{i=1}$, it performs the operations according to the rules
 - a. For $(w_i[x], l)$, a new version $x_{i,n}$ is created where n is the index of the operation, i.e., $(w_i[x_{i,n}], l)$ is done.
 - b. For $(r_i[x], l)$, if the index of the operation is $n > 0$, the value written by $(w_i[x_{i,n}], m)$ with index n is read. If the index is 0, $(r_i[x], l)$ reads from the last version of x written by a committed $T_{j=m}$ that has the largest timestamp $< ts(T_{j=1})$.
 - c. When all operations of $T_{i=1}$ have been performed, P_i locally commits $T_{i=1}$.
- II.3 For the $T_{i=1}$ that have been queued for later execution by II.1.b, P_i periodically checks whether $RS(T_{i=1}) \cap CS(T_{i=1}) = \emptyset$. If so then P_i initiates execution of $T_{i=1}$, provided that $T_{i=m}$ has been locally committed for all $m < i$ in SC. (A new timestamp is not issued).
- II.4 $P_{lub(i)}$ recognizes when every subtransaction of T_i has been locally committed, and then sends a Commit to Q .

Steps I.1, I.2, and I.3 require some level of trust. These steps deal directly with the multilevel transaction, so will see operations at various security levels. The amount of trusted code needed to implement these is very small relative to what would be required to construct a complete trusted scheduler. The remainder of the algorithm can be done with untrusted processes.

Since the Q_i deal only with entities of a single security level, they may be untrusted (for the access control policy here), so I.4 and I.5 can be performed without trusted processes.

For I.6, notice that the global commitment of a multilevel transaction is a technical requirement only, used solely to let the "user" know that the work submitted is finished. The rest of the scheduling mechanism does not make use of this information, but relies only on the behavior of the local schedulers.

II.1 can be untrusted since the computation required relies solely on information available at P_m for $m \leq 1$. The same observation applies to II.2 and II.3. II.4 clearly can be implemented untrusted.

6. VARIATIONS ON THE ALGORITHM

The version of the algorithm presented above is very pessimistic (conservative?) in that it waits to execute a subtransaction until it is absolutely certain that it can be executed without (locally) aborting or rolling back. Aside from minor changes that make the algorithm somewhat more optimistic, which may be beneficial for some applications, we believe that the algorithm is intuitively as good as can be done for these kinds of transactions.

Given the nature of multilevel transactions, the conventional definitions of database theory, and the security policy, there seems to be limited choice in the approach to constructing a multiversion timestamp scheduler if trusted processes are to be minimized.

There appear to be three general statements that can be made about this family of algorithms. First, because of the security policy, it seems necessary that there be some way of determining when the subtransaction are finished, so that it is safe to use the values they produce. This is our notion of local commit or abort. Second, because of the required atomicity of database transactions, if one local commit occurs, then all local commits must occur. Third, the behavior of higher security level operations must conform to what is done at lower levels in order to obtain correct results and prevent covert channels. The variations of the algorithm arise by enforcing these three conditions in slightly different ways.

For reflexive reads-froms, a Read of a data item x cannot occur until the proper Write operation has been done, possibly by a lower security level subtransaction. We have chosen to make the higher security level subtransactions wait until the lower level ones from the same multilevel transaction have locally committed to insure that the correct version is available to be read. A more optimistic alternative would allow the higher level subtransaction to begin and progress until some Read operation could not be performed because the corresponding

Write operation had not been completed. The higher level subtransaction could continue to attempt to read other data items until a Write operation is encountered (which could depend on a blocked Read operation for its value), at which point the whole subtransaction would be suspended. As the appropriate Write operations are done and the blocked Read operations completed, the subtransaction would resume. This technique requires significantly more overhead than the pessimistic approach, but may be warranted for some applications when it is unlikely that the reflexive reads-froms will result in waiting.

For transaction reads-froms, there are two variations, both more optimistic than the one presented. The first is similar to the technique for reflexive reads-froms. Subtransactions process their operations up to the point of reading data that might be invalidated by a Write operation at the same or lower security level, whence it is blocked until the lower level subtransaction from which it might read is locally committed. As before, such Read operations could continue until a Write operation is encountered, and then suspended. It could continue when the potentially offending lower level subtransaction is locally committed. The second variation is more optimistic in that it attempts to execute all subtransactions at the various security levels simultaneously. No initial determination of the read sets or write sets of potentially conflicting subtransactions is done. Rather, as subtransactions are completed at lower security levels, the resulting read sets and write sets are compared with the results already obtained at the higher levels. If the higher level actions are inconsistent with those of the lower levels, they must be rolled back and redone, reading the correct versions and redoing the Write operations that depend on them. These roll backs may involve cascading of these reversals through all higher security levels. Whether either of these variants is more appropriate than the pessimistic approach depends on the frequency of the need to invoke suspensions or rollbacks.

7. PROOF OF CORRECTNESS OF THE ALGORITHM

We must show that the multiversion (multilevel) history produced by the algorithm is **ISR** by showing that it is equivalent to a one-copy serial history. To this end let **G** be the multiversion history produced by arranging the multilevel transactions in timestamp order with operation indices and versions ordered so that **G** is one-copy serial. That this is possible is trivial. Let **H** be the multiversion history produced by the algorithm. Clearly **H** satisfies the definition of a multiversion history. We have the following result.

Theorem Let $T = \{T_1, T_2, \dots, T_n\}$ be a set of multilevel transactions. If **H** is a multiversion history produced by the algorithm for **T**, then **H** is **ISR**.

Proof Let G be as defined above. It is obvious from the definition of G and the specification of the algorithm, that G and H have the same reflexive reads-from relationships. Thus it is sufficient to show that G and H have the same transaction reads-from relationships to prove the theorem.

First, suppose T_j reads- x -from T_i in G so there are $(w_i[x], l) \in T_i$, $(r_j[x], m) \in T_j$ for which $(w_i[x], l) <_G (r_j[x], m)$, and no other Write operation on x falls in between. Suppose now that T_j does not read- x -from T_i in H . Then there is a $(w_k[x], l) \in T_k$ for which $(w_i[x], l) <_H (w_k[x], l) <_H (r_j[x], m)$. Now if $i=k$, then $(r_j[x], m)$ would not have read x from the last Write of x by a committed subtransaction as required by the algorithm, so i , j , and k must be distinct. Since $(w_i[x], l) <_H (w_k[x], l)$, the local scheduler P_1 must have scheduled $(w_k[x], l)$ after $(w_i[x], l)$, so $ts(T_i) = ts(T_{i|1}) < ts(T_{k|1}) = ts(T_k)$. Again, by the algorithm, P_m would not allow $T_{j|m}$ to begin until $T_{k|m}$ was locally committed, since it reads x after it, so $ts(T_k) = ts(T_{k|1}) < ts(T_{j|m}) = ts(T_j)$. Therefore T_k appears between T_i and T_j in G , contradicting that T_j reads- x -from T_i in G . Hence T_j does read- x -from T_i in H .

Conversely, suppose T_j reads- x -from T_i in H so there are $(w_i[x], l) \in T_i$, $(r_j[x], m) \in T_j$ for which $(w_i[x], l) <_H (r_j[x], m)$, and no other Write operation on x falls in between. Suppose now that T_j does not read- x -from T_i in G . Then there is a $(w_k[x], l) \in T_k$ for which $(w_i[x], l) <_G (w_k[x], l) <_G (r_j[x], m)$. If $i=k$, G would not be one-copy serial. As before, i , j , and k are all distinct, and $ts(T_i) = ts(T_{i|1}) < ts(T_{k|1}) = ts(T_k) < ts(T_{j|m}) = ts(T_j)$ because G is one-copy serial in timestamp order. Since $T_{k|1}$ writes x and has an earlier timestamp than $T_{j|m}$ but later than that of $T_{i|1}$, the algorithm would have finished $T_{k|1}$ before starting $T_{j|m}$, contradicting that T_j reads- x -from T_i in H . Thus T_j does read- x -from T_i in G .

We conclude that G and H have the same reads-from relationships and so are equivalent. Therefore H is 1SR. ■

8. CONCLUSION

The notions of atomicity for transaction processing that are usually suggested for databases are not easily reconcilable with those of multilevel secure systems. This is extremely problematic for multilevel secure database systems. Users' expectations may not be met if what the user considers a single transaction is decomposed into a sequence of single-level transactions that are then treated as non-communicating entities by the system's concurrency control mechanisms. Further, it is incumbent upon those who develop multilevel secure database systems to ensure that the users' needs and expectations are met to avoid misunderstandings about the system's functionality. To this end, we have

proposed the idea of multilevel transactions to resolve these difficulties. In cases where this is not an acceptable solution, system-high systems may be a solution, or developing completely trusted database systems, though this would be a significantly more costly route.

In this paper we have defined *multilevel transaction* for multilevel secure databases and defined a notion of correctness that is consistent with the traditional idea of correctness for database systems. To demonstrate the applicability of these ideas, an algorithm for correct transaction processing within this framework was presented for a multiversion architecture multilevel database. Very few trusted processes are needed to implement the algorithm, which greatly reduces the time and cost needed to develop a system using the algorithm.

We chose to develop the algorithm for the kernelized architecture since it has been the one of most interest to the database security community. The problem for multilevel secure database systems based on the replicated architecture [5], however, is no less interesting a research (and application) issue. An algorithm for this case, based on the correctness criterion for transaction processing in replicated database systems, will be the subject of future work.

9. REFERENCES

1. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," The Mitre Corp., March 1976.
2. P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
3. Oliver Costich, "Transaction Processing Using an Untrusted Scheduler in a Multilevel Database with Replicated Architecture", in *Database Security V: Status and Prospects*, ed. Sushil Jajodia and Carl Landwehr, North-Holland, 1992.
4. Oliver Costich and John McDermott, "A Multilevel Transaction Problem for Multilevel Secure Database Systems and its Solution for the Replicated Architecture", *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA May 1992.
5. Judith N. Froscher and Catherine Meadows, "Achieving a Trusted Database Management System using Parallelism," in *Database Security*

II:Status and Prospects, ed. Carl Landwehr, pp.151-160, North-Holland, 1989.

6. Sushil Jajodia and Boris Kogan, "Transaction Processing in Multilevel-Secure Databases using Replicated Architecture" in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 360-368, Oakland, CA May 1990.
7. T.F. Keefe and W.T. Tsai, "Multiversion Concurrency Control for Multilevel Secure Database Systems" in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 369-383, Oakland, CA May 1990.
8. William T. Maimone and Ira B. Greenberg, "Single-Level Multiversion Schedulers for Multilevel Secure Database Systems" in *Proceedings of the Sixth Annual Computer Security Applications Conference*, pp.137-147, Tucson, AZ December 1990.
9. John McDermott, Sushil Jajodia, and Ravi Sandhu, "A Single Level Scheduler for the Replicated Architecture for Multilevel-Secure Databases" in *Proceedings of the Seventh Annual Computer Security Applications Conference*, San Antonio, TX December 1991
10. Richard C. O'Brien, J.T. Haigh, and D.J. Thomsen, "Trusted Consistency Policy" Rome Air Development Center Technical Report RADC-TR-90-387, December 1990.
11. C.H. Papadimitriou, *The Theory of Concurrency Control*, Computer Science Press, 1986.

ORANGE LOCKING: CHANNEL-FREE DATABASE CONCURRENCY CONTROL VIA LOCKING

John McDermott
Naval Research Laboratory
Sushil Jajodia
George Mason University

The concurrency control lock (e.g. file lock, table lock) has long been used as a canonical example of a covert channel in a database system. Locking is a fundamental concurrency control technique used in many kinds of computer systems besides database systems. Locking is generally considered to be interfering and hence unsuitable for multilevel systems. In this paper we show how such locks can be used for concurrency control, without introducing covert channels.

1. Introduction

A database system is a software system that provides a collection of predefined operations with three features: 1) efficient management of large amounts of persistent data (the database), 2) transaction management for transactions composed of those operations on the data (concurrency control, atomicity, and recovery from failure), and 3) a data model that provides a simple abstraction for understanding how the predefined operations and data interact. Our concern is with the second of these features, transaction management.

Transaction management for conventional centralized database systems is fairly well understood and much progress is being made for distributed and federated database systems [14]. Our concern is with transaction management in what we call *multilevel database systems* (which may also be centralized, distributed, federated, etc.). Multilevel database systems assign their data to security classes and restrict database operations based on those classes [6]. The security classes are partially ordered; the data and operations are considered to be in various levels, hence the term multilevel.

A database system that just provides security classes and restrictions on operations is not multilevel. An additional feature of multilevel database systems is their ability to enforce the classes and restrictions in the face of nontrivial attempts to bypass or tamper with the enforcement mechanisms. One of the most difficult challenges for multilevel databases is the *covert channel* problem. A database system user with "low" privileges can obtain information from "higher" security classes by having it leaked into his or her security class from the higher classes, by a Trojan horse or virus, via a covert channel. A covert channel is a means of unauthorized interprocess communication that uses a mechanism ~~not intended for interprocess communication~~.

The concurrency control lock (e.g. file lock, table lock) has long been used as a canonical example of a covert channel in a database system. Locking is a fundamental concurrency control technique used in many kinds of computer systems besides database systems. Locking is generally considered to be interfering, in the sense of [5,13], and hence unsuitable for multilevel systems.

In this paper we show how locks can be used for concurrency control, without introducing covert channels. We have developed three locking algorithms that do not introduce signalling channels¹ yet they produce serializable transaction histories. Previous algorithms for concurrency control without channels have relied on timestamping [1,8,9] or are based on subtle properties of a particular database system architecture [3,7,11]. Our orange locking algorithms do not use timestamps and they do not depend on the underlying architecture of the database system. The rest of this paper is organized as follows. First, we discuss transactions, conventional locking, and covert channels. Then we present three locking algorithms: conservative orange locking, reset orange locking, and optimistic orange locking. We show these algorithms to be correct (serializable) and secure (noninterfering), and discuss their deadlock properties. Finally, because of our own interest in the replicated architecture, we show how orange locking can be used to implement immediate-write algorithms for the replicated architecture.

2. Transaction Management in Multilevel Databases

A transaction is an abstract unit of concurrent computation that executes atomically. The effects of a transaction do not interfere with other transactions that access the same data. Also, a transaction either happens with all of its effects made permanent or it doesn't happen and none of its effects are permanent. A useful model of a transaction must show how these properties can be achieved by composing smaller units of computation, when those smaller units are not necessarily guaranteed to compose into an atomic transaction. Thus the model must be concerned with showing potential conflicts between operations and with showing arbitrary orderings. Since we are managing transactions for secure database systems, our model must also reflect the security policy enforced by the DBS [10].

In this report we model transactions as sequences of abstract *read*, *read_lock*, *read_unlock*, *write*, *write_lock*, *write_unlock*, *commit*, and *abort* operations, denoted $r[x]$, $rl[x]$, $ru[x]$, $w[x]$, $wl[x]$, $wu[x]$, c , and a , respectively. The sequence models the order in which database operations are sent to the transaction management algorithms, without modeling the control structure of transactions themselves. Modeling transactions as sequences is desirable because the sequences can be used in a noninterference model [5,13] to reason about the security of our algorithms.

Definition 1. A database D is a finite set of pairwise disjoint data items that can be

1. We distinguish implementation invariant (i.e. inherent in the algorithm itself) covert channels as signalling channels. An implementation of a signalling-channel free algorithm may still have covert channels.

operated on by a single atomic database operation. Each data item x in D has a countable domain $dom(x)$. A *database state* is an element of the Cartesian product of the domains of elements of D , that is, a state associates a value with each data item in the database. The integrity constraints on a database specify a subset of the Cartesian product, that is, the consistent database states. A *transaction state* of a transaction is an element of the Cartesian product of the domains of the transaction's basis set. A transaction state associates a value with each data item that is read or written by the transaction. A database system state, or *DBS state*, on a history H is a tuple containing a database state as its first element and a transaction state for every transaction in history H . Each database operation maps a DBS state to a DBS state. \square

Definition 2. A *transaction* T is a finite sequence of database operations from some finite set O . Usually $O = \{ r[\bullet], rl[\bullet], ru[\bullet], w[\bullet], wl[\bullet], wu[\bullet], c, a \}$. We will take this read-write model as our definition unless we say otherwise. We denote single operations as one-element sequences thus $\langle r[x] \rangle$. Concatenation of sequences is denoted by juxtaposition. If transaction T reads x , then writes x and commits we can denote transaction T as the sequence $\langle r[x] \rangle \langle w[x] \rangle \langle c_T \rangle$. We can also show a transaction as a concatenation of unspecified sequences of database operations, like $T = \alpha \langle r[x] \rangle \beta$. We will always use lower case Greek letters to indicate subsequences of database operations. We use $\langle \rangle$ to denote the empty sequence.

If a sequence of database operations is a transaction, we further require that if the transaction $T = \alpha \langle p \rangle$, that is, the sequence of operations α followed the singleton sequence $\langle p \rangle$, then p must be either c_T or a_T and also that neither c_T nor a_T be in α . \square

Definition 3. Let S_1 be a sequence that contains only distinct elements and let S_2 be a sequence that contains only distinct elements. Say that these two sequences are *compatible* if they do not contain inconsistent orderings of elements common to S_1 and S_2 . For example, if $a, b \in S_1$ and $a, b \in S_2$ and if S_1 orders a and b as $a < b$ and S_2 orders a and b as $b < a$, then S_1 and S_2 are not compatible sequences. Let *image* S_1 denote the set of elements in sequence S_1 . Define the *shuffle* of two compatible sequences S_1 and S_2 , denoted $S_1 * S_2$, to be the set of all sequences that contain just the elements of $(image\ S_1) \cup (image\ S_2)$ and contain S_1 and S_2 as subsequences. The extension to the shuffle $S_1 * S_2 * \dots * S_k$ of more than two compatible sequences is straightforward. \square

Definition 4. A *history* H over a set of transactions $T = \{ T_1, T_2, \dots, T_k \}$ is an element of the shuffle of T , that is H is a sequence in $T_1 * T_2 * \dots * T_k$. A *serial history* has every operation of transaction T_i before every operation of transaction T_j (or vice versa), for every pair of transactions (T_i, T_j) in H . \square

Definition 5. We define two operations $p[x]$ and $q[x]$ to *conflict* if one of them is a write operation. Intuitively conflicting operations do not commute; we get different results if conflicting operations p and q are done in different orders. We say that two transactions conflict if they contain conflicting operations. \square

We can now define *conflict equivalence* using the notion of conflicting operations.

Definition 6. Two histories H_1 and H_2 are (conflict) equivalent if

1. they are defined over the same set of transactions and operations,
2. for any pair of conflicting operations $p_i[x]$, $q_j[x]$, (i not necessarily distinct from j) such that a_i, a_j are not in H_1 , we have
 $H_1 = \alpha_1 \langle p_i[x] \rangle \beta_1 \langle q_j[x] \rangle \gamma_1$ iff $H_2 = \alpha_2 \langle p_i[x] \rangle \beta_2 \langle q_j[x] \rangle \gamma_2$

□

In this discussion we take the view that histories are correct if they are serializable, that is, equivalent to some serial history. Because aborted transactions have no permanent effect on the database state we do not include them in our equivalent serial histories. Because active transactions (i.e. those that have not committed or aborted yet) may abort, we do not include them either. To accommodate this in our equivalence we define the *committed projection* $C(H)$ of a history H to be the history obtained from H by removing operations that belong to uncommitted transactions.

Definition 7. Formally, we say that history H is *serializable* if its committed projection $C(H)$ is equivalent to some serial history. □

Definition 8. A *serialization order* of a history H is the order that transactions appear in a serial history that is equivalent to the committed projection of history H . A serial history equivalent to the committed projection of H is not necessarily unique so H may have several serialization orders. □

Definition 9. To model the security policy enforced by our database systems we introduce a finite set of *subjects* S , and a finite lattice (SC, \leq) of *security classes*. The data items in D of our transaction model will be the passive entities of our security model, that is, abstract units of protected computer resources. A subject is an abstract unit of secure computation. We relate subjects to transactions by defining a subject to be a sequence of database operations. A subject may or may not be a transaction. Every transaction will be a sequence of one or more subjects and every subject in our security model will be in one and only one transaction. We may use the terms *higher* and *lower* to refer to a relation between two or more security classes. By *higher*, we mean strictly greater than and by *lower* we mean strictly less than. We use a mapping $\lambda: D \cup S \rightarrow SC$ to give the security class of every data element and every transaction.

The algorithms we present here apply to database systems that enforce the following security policy:

1. All transactions are single-level. That is, every subject in a transaction has the same security class and we can meaningfully apply our level function to transactions, thus $\lambda(T)$.
2. Subject S is not allowed to read data element $x \in D$ unless $\lambda(S) \geq \lambda(x)$.
3. Subject S is not allowed to write into data element $x \in D$ unless $\lambda(S) = \lambda(x)$.
4. $\lambda(S)$ and $\lambda(x)$ do not change.

□

Definition 10. If two transactions T_i and T_j have security classes $\lambda(T_i), \lambda(T_j)$ such that $\lambda(T_i) < \lambda(T_j)$, then T_i and T_j are *low* and *high* transactions with respect to each other. We introduce this definition simply for convenience of exposition. \square

Definition 11. A transaction T_i reads x from transaction T_j in history H if

$$H = \alpha \langle w_j[x] \rangle \beta \langle r_i[x] \rangle \gamma$$

and $\alpha_j \notin \beta$ and if $w_k[x] \in \beta$ then $\alpha_k \in \beta$. \square

Definition 12. A *read-down* is a read operation $r[x]$ of a transaction T_i such that $\lambda(T_i) > \lambda(x)$. Data item x is a *read-down data item*. If transaction T_i reads x from transaction T_j and x is a read-down data item, then T_i reads- x -down from T_j , and if for some data item x , T_i reads- x -down from T_j , then T_i reads-down from T_j . \square

3. Locking and Channels

Concurrency control via conventional locking is based on the following principle: 1) each operation that is to be scheduled includes a (possibly implicit) lock request, and 2) if a transaction requests a lock $pl_i[x]$ that conflicts with a lock $ql_j[y]$ that is already set then the requesting transaction is delayed. Two locks $pl_i[x]$ and $ql_j[y]$ conflict if their corresponding operations p and q conflict, $x=y$, and $i \neq j$. Locks can be implemented as a lock table inside the scheduler. Our abstract read lock $rl_i[x]$ is implemented as a lock table entry $\langle x, read, i \rangle$. Transactions that are delayed can be placed on a queue associated with the entry; the mechanism for effecting the delay depends on the underlying operating system. The setting and releasing of locks and the scheduling of operations is done by the scheduler. Transactions request operations and the scheduler returns the results when they are available.

Intuitively, locking should be sufficient by itself to ensure correct database system operation. Unfortunately, it is not. Locking intended to achieve serializability must also be *two-phase*, in the following sense. Transactions that use two-phase locking have a *growing phase* wherein all of a transaction's locks are set and a *shrinking phase* wherein all of its locks are released. A transaction's locks are not necessarily set or released all at the same time, but no lock may be set after a lock has been released. Formally, we say that for any data items x and y , and any transaction T_i , it is always the case that $pl_i[x]$ precedes $qu_i[y]$.

To make recovery from failures tractable, two-phase locking algorithms are often designed to be *strict*. A transaction scheduled by a strict two-phase locking algorithm holds all of its write locks until the end of the transaction, and then releases them together.

Conventional locking introduces a signalling channel. If a virus or Trojan horse in transaction T_i wishes to signal information to a less privileged transaction T_j (i.e. T_j runs in a lower security class) it can do so by **reading down**, from some predetermined data item x such that the security class of x is the same as T_j 's security class. Transaction T_i 's read request will set a read lock $rl_i[x]$. If transaction T_j now tries to

write into data item x , transaction T_j will be delayed by the read lock $rl_i[x]$. By selectively read locking and read unlocking data item x , transaction T_i can leak information to T_j . It is this well-known scenario we wish to prevent.

4. Optimistic Orange Locking (OOL)

Now we show how to use locks for concurrency control, in a way that does not introduce signalling channels. Our first algorithm is optimistic, that is, operations are never delayed by the scheduling algorithm. Instead, when a transaction is ready to commit, the scheduling algorithm checks the schedule to see if it is correct. If not, then some transaction is aborted (to be rerun later) to make the schedule correct.

In our first approach we simply let the high transaction T_j set read locks on low data items as in a conventional, untrusted database system. If a low transaction T_i then tries to set a write lock on one of the same data items, we immediately grant T_i 's write lock and change T_j 's read-down lock to an *orange lock*, indicating the possibility of an incorrect read.

Low transactions will not be interfered with by high transactions following this approach. However, we have to decide what to do with high transactions that read data via orange locks instead of read-down locks. If we simply inform the transactions of the orange locks but let them read anyway, the transactions will probably be incorrect. The read-down operations will have been invalidated by the conflicting write that was performed in a nonserializable fashion.

We can obtain serializable schedules by simply aborting a transaction whenever its first read down is orange locked. If most transactions only read down on a few data items and transactions are easy to restart, this approach will allow us to correctly schedule them in a simple manner. This approach begins to have problems when the number of read-downs increases or the cost of restarting a transaction is high. We can do better, at the expense of an increase in complexity, by reducing the number of aborts and making restarts easier.

First, we add a *local workspace* for each transaction. The local workspace contains storage for all the values a transaction will read down. We begin each transaction by having it perform all of its read-downs before beginning any processing. After all of the data items in the local workspace are read, the transaction proceeds as a conventional transaction, reading from and writing to the database directly, within the transaction's security class. Any read-downs during processing are performed from the transaction's local workspace.

Definition 13. A transaction T_i has a *home-free point* that it must reach before completing its processing. A transaction T_i has reached its home-free point when all data items x to be read down by T_i are either read locked and read into T_i 's local workspace or orange locked and read into T_i 's local workspace. \square

If a high transaction T_j reaches its home-free point without any orange locks it is allowed to proceed. If T_j has an orange lock set before it reaches its home free point, it

is aborted. This abort can be made a *lightweight abort*, that is, we do not need to re-submit the transaction to the scheduler. Instead we can abort by releasing all read-down locks, resetting the local workspace, and moving the transaction's program counter back to the beginning of the transaction. Thus we achieve the effect of a full abort with less overhead. Because we do all our read-downs together, we reduce the length of time we are likely to be interfered with by a low transaction.

Our simple optimistic approach can be shown to be correct because its histories are identical to histories produced by conventional two-phase locking. Transactions that do not conform to the conventional two-phase model are aborted and do not appear in the committed prefix that defines the current stable database state. Our workspace-based improvement is also correct; we will show how later in this paper.

The advantage of this optimistic approach is that we have a relatively simple algorithm, even with our workspace version. We do not have to change our conventional two-phase locking implementation very much. Unfortunately, we get poor performance if lock contention is heavy and we can also get starvation as a high transaction's read-down locks are repeatedly set to orange, forcing the high transaction to restart.

Remark. The potential for infinite overtaking suggests a possibility for denial of service. While this is theoretically true, it is of no practical concern. A more effective denial of service attack can be mounted with crude techniques such as resource exhaustion.

5. Conservative Orange Locking (COL)

If aborts and restarts are too expensive, but we still want to maintain correctness for our transactions, we can do so by using a conservative approach. Our approach is not conservative in the usual sense because it can still deadlock. Our approach is conservative in the sense that it does not need to abort any transactions for concurrency control reasons and also because it tries to avoid any possible missteps in its approach to scheduling.

In the OOL scheduler, we lock and schedule operations as we normally would, except we cannot delay low write operations to ensure correct read-down operations. In OOL we give up on the high transaction as soon as we detect a conflict with a low write operation. We can do better than this by trying to save the high transaction instead of aborting it. In the conservative orange locking approach, we will use the orange locks to identify a current low transaction that we can safely read from, thus we do not have to give up if a low transaction has a conflict with a read-down. To do this, we may have to resubmit some read-down operations that were invalidated by low transactions. In fact, we can sometimes do even better than rereading invalidated read-downs. If we override a read-down lock into an orange lock *before* we schedule the associated read-down operation, we can delay that read operation until we have identified the proper low transaction to read from. We will avoid performing an invalid read in the first place.

We continue with some data structure definitions. We will unavoidably use some terms before they are defined; we ask the reader to trust that all meanings will be resolved as quickly as possible.

5.1 Local Workspace

Each transaction has a local workspace that is used to hold the values of data items the transaction needs to read-down. The local workspace is used in the same way as in optimistic orange locking; any read-downs during processing are performed from the transaction's local workspace. In conservative orange locking, each read-down data item in the local workspace can be marked *read* or *unread*. These markers are used to determine when a transaction has reached its home-free point. Since a high transaction does not give up when it finds one of its read-down operations has been invalidated, the transaction must know which data items to reread or delay on in order to get a valid view of the database.

5.2 Read-Down Queue

The scheduler associates every transaction with a transaction-specific queue Q_i , called a *read-down queue*. Whenever a high transaction T_j must repeat or defer one of its reads, it does so in order to read from a currently active low transaction T_i . To do this, the scheduler places transaction T_j on the low transaction T_i 's read-down queue to wait for T_i to write the necessary value. Management of the read-down queues is done by the scheduler. A low transaction T_i is not even aware of the existence of its corresponding read-down queue Q_i .

Along with low transaction T_i 's read-down queue, the scheduler keeps a list W_i of values written by the corresponding transaction. For efficiency, this list W_i may be incorporated into the database system's cache and recovery log, depending on their implementation. When low transaction T_i commits, the scheduler services the reads requested by any high transaction T_j that was placed on T_i 's read-down queue. The values returned are taken from list W_i . (To preserve recoverability, cascadelessness, and strictness, the scheduler should not make orange locked data items in W_i available for reading via Q_i until transaction T_i has committed.)

5.3 Conservative Orange Locks: Overriding Read Locks for Read-Downs

The heart of our conservative approach is the way we use orange locks. Instead of passively marking data items, our orange locks actively affect the individual scheduling of reads and writes. Whenever a low transaction T_i needs to obtain a write lock on a data item x that is being read by a high transaction T_j , the scheduler tries¹ to *override* the high transaction T_j 's read-down of x . On behalf of transaction T_j , the scheduler converts T_j 's read lock to an orange lock on data item x . Whenever a data item x is orange locked on behalf of transaction T_j , data item x in T_j 's local workspace is also marked *unread* by the scheduler. Even if data item x had previously been read it is still marked *unread*. A high transaction T_j that has its read lock converted to an orange lock is placed on the appropriate read-down queue to wait for the overriding

1. The read lock may be released before it is overridden.

low transaction T_i to complete. At the same time, all of the read-down data items in T_i 's write set are also orange locked and thus marked *unread*. At this point we say that transaction T_j is *orange locked into transaction T_i* . If T_i commits then T_j will read-down from transaction T_i every data item in the write set of transaction T_i that T_j reads. If this happens then the override is considered to have occurred. If instead transaction T_i aborts, then all of the orange locks that were associated with it must be reset to read locks (the affected data items will all still be unread) and transaction T_j must continue to try to reach its read-down point. If another low transaction T_k tries to write lock data item x and high transaction T_j already holds an orange lock on x then the original orange lock is retained but the low transaction T_k gets its write lock and continues. We state this formally as the *orange locking rule*.

Definition 14. We denote the read set and write set of a transaction T_i as R_i and W_i respectively. We also define the *read-down set of transaction T_i* as the set E_i of all T_i 's read-down data items and we also define the *orange-locked set O_{ij}* as the set of all read-down data items that T_i reads down from transaction T_j via an orange lock. If transaction T_i reads x down and transaction T_j converts T_i 's read lock on x to an orange lock then $O_{ij} = E_i \cap W_j$. If transaction T_i reads x down and its read lock is not converted but x is in W_j then O_{ij} is empty. We will refer to this condition as the *conservative orange locking rule*, that is if data item x is in $E_i \cap W_j$, then $O_{ij} = E_i \cap W_j$ or $O_{ij} = \emptyset$.

5.4 The Conservative Orange Locking Algorithm

Now that we have a clear definition of the override operation, it is possible to talk about how orange locking is used in the algorithm. We give the steps to be followed by a transaction T_i and by the scheduler in serializing T_i 's operations.

- (1) Transaction T_i declares its read-down set E_i and its write set W_i .
- (2) The scheduler marks all of T_i 's local workspace *unread* and sets Q_i , its read-down queue, to empty.
- (3) While some read-down data item in its local workspace is still marked *unread*, transaction T_i submits read-down operations for those unread data items. If the read-down data item is read locked it is read from the database and marked *read* in the local workspace. Otherwise the data item must be orange locked and the transaction reads, via the scheduler's list W_j , from the committed transaction T_j whose write operation required conversion of T_i 's read lock into an orange lock. When this step completes, transaction T_i has reached its home-free point.
- (4) Transaction T_i now releases the locks on its read-down data items. The read-down locks can be released together in a single operation. Alternatively, if read-down locks are not released together and some low transaction T_j requests a write lock after T_i has reached its home-free point but before the scheduler has released all of T_i 's read-down locks, the scheduler **simply grants the write lock** and schedules the write before releasing the rest of T_i 's locks.

(5) Transaction T_i now performs the rest of its processing using conventional strict two-phase locking on data items within its own security class. If transaction T_i needs to perform a write operation on data item x at the same time another transaction T_j needs to read-down x , then T_i will override T_j 's read-down by converting T_j 's read-lock to an orange lock.

(6) When transaction T_i commits, all of the high transactions that are waiting for T_i on the scheduler's queue Q_i are allowed to read from T_i , via the scheduler's list W_i . At this point transaction T_i will have succeeded in overriding the reads of those high-er transactions, thus requiring them to read from list W_i .

□

A COL scheduler avoids starvation because it selects a specific active low transaction for a high transaction to read from, or it schedules the high transaction to read from the database itself via valid read-downs. It achieves this at the expense of complexity of mechanism. Note that by waiting until the selected low transaction completes, we incur less delay than our intuition would suggest, since we would have had to wait almost as long for the selected low transaction if it had already held the lock.

The serialization order established by a COL scheduler is determined by the home-free points between security classes and by the lock points within the same security class. The home-free point of a transaction must come either before or after the lock point of every conflicting transaction. By holding its read-down locks until its home-free point, conservative orange locking becomes a four-phase protocol. There is a growing and a shrinking phase for read-downs and then a growing and a shrinking phase for intra-class reading and writing.

6. Reset Orange Locking (ROL)

Intuitively, the conservative orange locking rule may seem to be too strong. We would like to do something less than orange lock the entire intersection of a transaction's read-down set and the corresponding update transaction's write set. Fortunately, we can do better than the conservative orange locking rule, if we are willing to return to the possibility of infinite overtaking or starvation. We can do this while still avoiding the need to abort any high transactions that have had read-downs invalidated by low write operations.

In the reset orange locking algorithm, we use the same definitions. Again the local workspace only holds the values the transaction needs to read down. In the ROL algorithm, values to be written are not held in a list W_i by the scheduler and there is no read-down queue Q_i for a transaction. Instead, we can let transactions read down directly from the database.

6.1 Reset Orange Locks: Resetting Read Locks for Read-Downs

In reset **orange** locking, just as in COL, low transactions override the read down locks of high transactions. Whenever a low transaction T_i needs to obtain a write

lock on a data item x that is being read by a high transaction T_j , the scheduler tries to override the high transaction T_j 's read-down of x .

In reset orange locking, the effect of an override is different from COL. First, the scheduler sets low transaction T_i 's write lock on data item x and schedules transaction T_i 's write operation. Then the scheduler marks data item x in transaction T_j 's local workspace as unread. Next, the scheduler releases high transaction T_j 's read lock. Eventually transaction T_j requests the scheduler to set it again, by asking for the corresponding read operation. The result of this attempt is that high transaction T_j 's read request is queued waiting for a chance to read according to the normal rules of two-phase locking (e.g. it may have to wait for other writes besides T_i 's). In the case of reset orange locking, if another low transaction T_k tries to write lock data item x and high transaction T_j has once more obtained its read lock on data item x , the new low transaction *does* override high transaction T_j . This repeated overriding can cause starvation and transaction T_j may never reach its home-free point.

Because a transaction holds all its read-down locks until it reaches its home-free point it is sure to detect (via resetting) any writes that could potentially invalidate a previous or pending read operation.

6.2 The Reset Orange Locking Algorithm

We give the steps followed by a transaction T_i scheduled by ROLand by the ROL scheduler. We follow the same style of exposition to allow comparison with COL.

- (1) The scheduler marks all of T_i 's local workspace *unread*.
- (2) While some read-down data item in its local workspace is still marked *unread*, transaction T_i submits read-down operations for those unread data items. If the read-down data item is read locked it is read from the database and marked *read* in the local workspace. Otherwise transaction T_i 's read request is queued waiting for a chance to read according to the normal rules of two-phase locking. When this step completes, transaction T_i has reached its home-free point.
- (3) Transaction T_i now releases the locks on its read-down data items. The read-down locks can be released together in a single operation. Alternatively, if read-down locks are not released together and some low transaction T_j requests a write lock after T_i has reached its home-free point but before the scheduler has released all of T_i 's read-down locks, the scheduler simply grants the write lock and schedules the write before releasing the rest of T_i 's locks.
- (4) Transaction T_i now performs the rest of its processing using conventional strict two-phase locking on data items within its own security class. If transaction T_i needs to perform a write operation on data item x at the same time another transaction T_j needs to read-down x , then T_i will override (via the scheduler) T_j 's read-down by converting T_j 's read-lock to a queued read-lock request. Transaction T_i commits according to the rules of conventional strict two-phase locking.

□

The reset orange locking algorithm is simpler than conservative orange locking. It also does not require declaration of read-down and write sets. In return for this decrease in complexity, we now have the possibility of starvation.

7. Correctness

Our proofs depend on the following important definition:

Definition 15. We define the home-free point HFP_i of transaction T_i to be the first unlock operation $ru_i[x]$ performed on a data item x that is read down by T_i . We define the lock point LP_i of transaction T_i to be the first unlock operation $qu_i[y]$ performed on a data item y in the same security class as transaction T_i . Intuitively, the lock point of T_i is the conventional lock point associated with strict two-phase locking, as we use it within a security class. For a transaction that does not read down we consider the home-free point to be the lock point. \square

We will now show the correctness of ROL. Instead of the usual graph theoretic proof, we argue directly towards the definition of conflict serializability.

Given any history H produced by an ROL scheduler, construct from H a serial history H_s as follows: take the committed projection of H and for every pair of transactions (T_i, T_j) in $C(H)$, if

1. the security class of transaction T_i is the same as the security class of transaction T_j , that is, $\lambda(T_i) = \lambda(T_j)$, then put the transactions into H_s in the order that their lock points appear in $C(H)$, or
2. $\lambda(T_i) < \lambda(T_j)$ or vice versa, then put the transactions in H_s in the order that the home-free point of the high transaction appears with respect to the low transaction's lock point in $C(H)$, or
3. some transaction is pairwise incomparable to every other transaction in $C(H)$; put each such transaction at the end of H_s .

The serial history H_s is defined over the same set of transactions and has the same set of operations as $C(H)$. We show that the committed projection $C(H)$ orders pairs of conflicting operations the same as serial history H_s by constructing a chain of equivalent histories starting from $C(H)$ and ending with H_s .

By the definition of conflicting operation and conflict equivalence, we can swap two adjacent nonconflicting operations in a history and the result will be equivalent to the original history. Thus, if the conflicting operations in $C(H)$ and H_s are already in the same order we can transform $C(H)$ into H_s via a finite number of equivalence preserving swaps.

To show that all pairs of conflicting operations are already in the same order, we consider three cases:

Case $\lambda(T_i) = \lambda(T_j)$: Only operations on data items at the same security class conflict, all other operations must be read downs. By the strict two-phase locking of step (4) we know that, for any pair of conflicting operations $p_i[x]$, $q_j[x]$, either

$$(1) C(H) = \alpha \langle p_i[x] \rangle \beta \langle LP_i \rangle \gamma \langle q_j[x] \rangle \delta \text{ and}$$

$$(2) H_s = \epsilon \langle p_i[x] \rangle \zeta \langle LP_i \rangle \eta \langle q_j[x] \rangle \theta$$

or vice versa, depending on which lock point comes first in $C(H)$. Thus all pairs of conflicting operations from transactions in the same security class are ordered the same in $C(H)$ and H_s .

Case $\lambda(T_i) < \lambda(T_j)$: Definition 9 tells us that we can only have conflicting pairs of the form $w_i[x]$, $r_j[x]$, in either order. Suppose we have committed projection

$$(3) C(H) = \alpha \langle r_j[x] \rangle \beta \langle w_i[x] \rangle \gamma$$

for some pair of operations $w_i[x]$, $r_j[x]$. By steps (2), (3), and (4) of the algorithm, the committed projection must be

$$(4) C(H) = \alpha \langle r_j[x] \rangle \delta \langle HFP_j \rangle \epsilon \langle w_i[x] \rangle \zeta \langle LP_i \rangle \eta$$

We know that every other pair of operations $w_i[y]$, $r_j[y]$ in $C(H)$ must also be shuffled such that

1. $r_j[y] \in \alpha$ or $r_j[y] \in \delta$, by the definition of home-free point, and
2. $w_i[x] \notin \alpha$ and $w_i[x] \notin \delta$, since transaction T_j will be in its step 2 or step 3 and transaction T_i will be in its step 4.

Thus if one pair of conflicting operations $w_i[x]$, $r_j[x]$ is ordered according to equation (3) then all pairs of conflicting operations are also ordered the same way, which corresponds to the application of serial history construction rule 2: place transaction T_j before transaction T_i in H_s because T_j 's home-free point HFP_j is before T_i 's lock point LP_i in $C(H)$.

Suppose that the committed projection has some pair of operations $w_i[x]$, $r_j[x]$ in the other order, that is

$$(5) C(H) = \alpha \langle w_i[x] \rangle \beta \langle r_j[x] \rangle \gamma$$

By steps (2), (3), and (4) of the algorithm, the committed projection must be

$$(6) C(H) = \alpha \langle w_i[x] \rangle \delta \langle LP_i \rangle \epsilon \langle r_j[x] \rangle \zeta \langle HFP_j \rangle \eta$$

We know that every other pair of operations $w_i[y]$, $r_j[y]$ in $C(H)$ must also be shuffled such that

1. $r_j[y] \notin \alpha$ and $r_j[y] \notin \delta$, since transaction T_j will be in its step 2 or step 3 and transaction T_i will be in its step 4, and
2. $w_i[x] \in \alpha$ or $w_i[x] \in \delta$, by the definition of two-phase locking.

Thus if one pair of conflicting operations $w_i[x], r_j[x]$ is ordered according to equation (5) then all pairs of conflicting operations are also ordered the same way, which corresponds to the application of serial history construction rule 2: place transaction T_i before transaction T_j in H_s because T_i 's lock point LP_i is before T_j 's home-free point HFP_j in $C(H)$.

Case $\lambda(T_i) > \lambda(T_j)$: The arguments are symmetric to the preceding case.

□

The correctness proofs for COL and workspace-based OOL are very similar. For simple OOL we simply note that any potentially nonserializable transactions are missing from $C(H)$ and use the proof for conventional two-phase locking given in [2].

8. Security

We argue informally that orange locking is noninterfering. We need to do this because some parts of the algorithm may be implemented as trusted code. We can restrict our discussion to read-down operations because they are the only parts of the algorithms that have the potential to affect the low state of the database system in a way that is interfering. We assume without discussion that no low state variables are changed explicitly by any of the algorithms, including error messages that might report a data item as being locked. Instead we are concerned with delays; that the value of time as a state variable can be made to change according to high inputs (read down requests) to our algorithms. In all three algorithms, if no write is requested while a read-down lock is set, there is no delay possible.

In COL scheduling we must set the low transaction's write lock immediately, before we invalidate the read-down of the high transaction. Likewise, the scheduling of the write operation must precede the orange locking action. If our mechanism for recording values in the list W_i causes a perceptible delay in returning an acknowledgment for the write, then COL scheduling could have a problem. However, the value of a write is usually recorded in a cache or recovery log or both, as part of the normal write process. If not, we can simply make the write operation always put the value in W_i , thus it becomes a constant time operation. We also need to make the action of placing a high transaction on the read-down queue part of the action of setting a write lock.

In ROL scheduling, we must also set the write lock immediately and schedule the write operation right away so as to make the write a constant time operation. The release and resetting of the high transaction's read lock will not interfere with any low transaction. Also, in ROL we do not have to deal with read-down queues and lists of writes, so it is easier to make ROL secure.

OOL scheduling is trivially noninterfering; the scheduler only has to override read-down locks. Since the orange locked transaction will be aborted, there is no problem of getting correct values for the read that is overridden.

9. Deadlocks

Conventional two-phase locking is subject to deadlocks. Two or more transactions can obtain exclusive locks (i.e. write locks) that the others are waiting for and none of the transactions will be able to proceed. This is because the two-phase nature of the algorithm precludes releasing some lock and resetting it later. Deadlocks are usually resolved by restarting one of the transactions involved in the deadlock.

Orange locking has the same potential for deadlocks, within transactions at the same security class, for the same reason. Read-down operations across security classes cannot cause deadlocks at lower classes because read locks can never delay a lower transaction. Transactions that read down via orange locking can be involved in deadlocks because they may also interact with transactions in their own security class.

10. Application to the Replicated Architecture

While orange locking is applicable to kernelized multilevel database systems we are interested in its potential for use in concurrency and replica control for the replicated architecture [4].

The frontend-backend architecture with full replication has been around as a concept for some time [12]. In its SINTRA¹ project, the Naval Research Laboratory is currently prototyping several frontend-backend architectures with full replication. What will be called in this paper the SINTRA architecture was proposed by Froscher and Meadows.

The SINTRA architecture uses full replication to provide multilevel security. There is an untrusted backend database system for each security class. Data from dominated security classes is replicated in each backend system. Logically, the user is allowed to read down and write at the same class but physically the frontend reads all data at the same class and writes at the same class and up into dominating classes to maintain the replicas. It is important to remember that while the replicated architecture uses distributed database system technology, the replicated approach is a centralized architecture. These techniques may be adapted to distributed database systems but not without careful consideration of additional issues. Figure 1 illus-

1. Secure INformation Through Replicated Architecture.

trates the basic replicated architecture, for the partial order of Figure 2. Notice that the low data appears at all backends, left data at the left and high backends, etc.

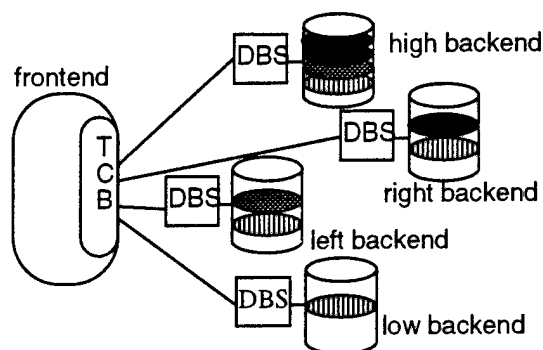


Figure 1. The Replicated Architecture

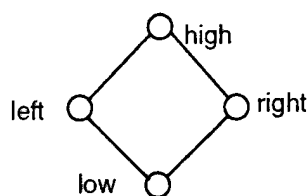


Figure 2. Partial Order for Figure 1

Several *deferred-write* algorithms have been developed for the replicated architecture [3,7,11]. Deferred-write replica control algorithms perform updates on one replica at the time the update is requested and defer the other updates until later. In contrast, *immediate-write* algorithms update all replicas simultaneously.

Immediate-write concurrency control algorithms for the replicated architecture require the concurrency control mechanism in general and the lock table in particular to reside on the frontend. Obtaining a lock on data item x must lock all replicas of x , simultaneously, by a global lock for x acquired on the frontend. This is because the write operations must be sent to the backend databases simultaneously. Since orange locking can provide this kind of concurrency control without introducing signalling channels, it is suitable for immediate-write concurrency control in the replicated architecture.

11. Conclusions

Locking is the preferred mechanism for achieving concurrency control in practical systems. Conventional locking introduces signalling channels. We have shown three different ways of provided channel-free locking for concurrency control: conservative orange locking, reset orange locking, and optimistic orange locking.

The structural differences between these three algorithms are significant. The COL approach avoids the possibility of starvation in the theoretical sense. In the practical

sense, it also avoids multiple overrides that could reduce the performance of ROL. COL is structurally more complex than the other two approaches, in both algorithm and data structures. The ROL approach is simpler than COL in algorithm and data structure. It can suffer from multiple overrides of its read locks but it does not need to abort transactions to deal with overrides. The OOL approach without the local workspace has the simplest structure of all. In systems where conflicts are few, this simplicity will give it the best performance of three. As the level of conflict (number of conflicting operations per transaction) and multiprogramming (number of transactions active at the same time) increases, determining best performance among the three approaches becomes problematic.

The need to declare read-down sets and write sets in COL is not as limiting as it first seems. The declarations prohibit correct scheduling of ad-hoc transactions with COL, but not interactive applications. Many interactive DBS applications are supported through forms, which are compatible with COL scheduling.

Some trusted code may be necessary to implement orange locking. The lock tables themselves may be multilevel objects and should only be accessed by trusted code. How much trusted code is required outside the lock table is also problematic. In some systems it may be possible to implement the lock table as a collection of single-level objects and the lock manager as a collection of single-level processes.

Future work should investigate performance issues in greater depth and look into orange locking implementation architectures with minimal trusted code. Extension of four-phase locking to untrusted systems, with an eye to increasing concurrency, is something else we plan to investigate.

Acknowledgments

We would like to thank Ravi Sandhu and the students in his *Advanced Topics in Computer Security* for their animated discussion of this problem, Oliver Costich for naming the home-free point, and Myong Kang, Judy Froscher, and the anonymous referees for their comments.

References

1. P. Amman and S. Jajodia, "A Timestamp Ordering Algorithm for Secure, Single-Version, Multilevel Databases", in *Database Security V: Status and Prospects*, ed. C. E. Landwehr and S. Jajodia, North-Holland, Amsterdam, 1992
2. P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987, ISBN 0-201-10715-5.
3. O. Costich, "Transaction Processing Using an Untrusted Scheduler in a Multilevel Database with Replicated Architecture", in *Database Security V: Status and Prospects*, ed. C. E. Landwehr and S. Jajodia, North-Holland, Amsterdam, 1992.

4. J. Froscher and C. Meadows, "Achieving a trusted database management system using parallelism", in *Database Security II: Status and Prospects*, ed. C. E. Landwehr, North-Holland, Amsterdam, 1989, ISBN 0-444-87483-6, pp. 253-261.
5. J. Gougen and J. Meseguer, "Unwinding and Inference Control", *Proceedings of 1984 IEEE Symp. on Security and Privacy*, Oakland, CA. pp. 75-86.
6. T. Hinke, "DBMS Technology vs. Threats", in *Database Security: Status and Prospects*, ed. C. E. Landwehr, North-Holland, Amsterdam, 1988, pp. 57-87.
7. S. Jajodia and B. Kogan, "Transaction Processing in Multilevel-Secure Databases Using Replicated Architecture", *Proceedings of 1990 IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 360-368.
8. T. Keefe, W. Tsai, J. Srivastava, "Multilevel Secure Database Concurrency Control", *Proceedings of Sixth International Conference on Data Engineering*, Los Angeles, CA, February 1990, pp. 337-344.
9. W. Maimone and I. Greenberg, "Single-Level Multiversion Schedulers for Multilevel Secure Database Systems", *Proceedings of Sixth Annual Computer Security Applications Conference*, Tucson, AZ, December, 1990, pp. 137-147.
10. J. McDermott, O. Costich, M. Kang, "A Formal Model of Secure Transaction Management", *NRL TM 5540-192*, July, 1992.
11. J. McDermott, S. Jajodia, and R. Sandhu, "A Single-level Scheduler for the Replicated Architecture for Multilevel-Secure Databases", *Proceedings of Seventh Annual Computer Security Applications Conference*, San Antonio, TX, 1991, pp. 2-11.
12. "Multilevel Data Management", Committee on Multilevel Data Management, Air Force Studies Board, National Research Council, Washington, DC, 1983.
13. I. S. Moskowitz and O. L. Costich, "A Classical Automata Approach to Noninterference Type Problems", Franconia.
14. A. P. Sheth and J.A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, 22, 3 (Sep.), 1990, pp 183-236.

A Practical Transaction Model and Untrusted Transaction Manager for a Multilevel-Secure Database System

Myong H. Kang and Judith N. Froscher

Center for Secure Information Technology
Naval Research Laboratory
Washington, D.C. 20375

Oliver Costich †

Center for Secure Information Systems
George Mason University
Fairfax, Virginia 22030

Abstract

A new transaction model for multilevel-secure databases which use the replicated architecture is presented. A basic concurrency control algorithm and two variations are given based on this transaction model. We also present new correctness criteria for multilevel-secure databases which use the replicated architecture. Based on this criteria, we prove that our algorithms are correct.

† Supported by the Naval Research Laboratory under contract N0001489-C-2389.

1. INTRODUCTION

There are several approaches for multilevel database systems which protect classified information from unauthorized users based on the classification of the data and the clearances of the users. One, the integrity lock approach [5], attempts to combine encryption techniques with off-the-shelf database management systems. The trusted frontend applies an encrypted check sum to data in untrusted backend databases. Another, the kernelized approach [11], relies on decomposing the multilevel database into single level databases which are stored separately, under the control of a security kernel enforcing a mandatory access control policy.

The integrity lock approach is computationally intensive and has a potential covert channel. The kernelized approach can yield reduced performance due to the need for recombining single level data to produce multilevel data. Motivated by performance concerns, a replicated architecture approach has been proposed.

The replicated architecture approach [6] uses a physically distinct backend database management system for each security level. Each backend database contains information at a given security level and all data from lower security levels. The system security is assured by a trusted frontend which permits a user to access only the backend database system which matches his/her security level.

The SINTRA¹ database system, which is currently being prototyped at the Naval Research Laboratory, is a multilevel trusted database management system based on this replicated architecture. The replicated architecture system contains a separate database system for each security level. The database at each security level contains data at its own security level, and replicated data from lower security levels.

The SINTRA database system consists of one trusted front end (TFE) and several untrusted backend database systems (UBD). The role of the TFE includes user authentication, directing user queries to the backend, maintaining data consistency among backends, etc. Each UBD can be any commercial off-the-shelf database system. Figure 1 illustrates the SINTRA architecture.

1. Secure INformation Through Replicated Architecture

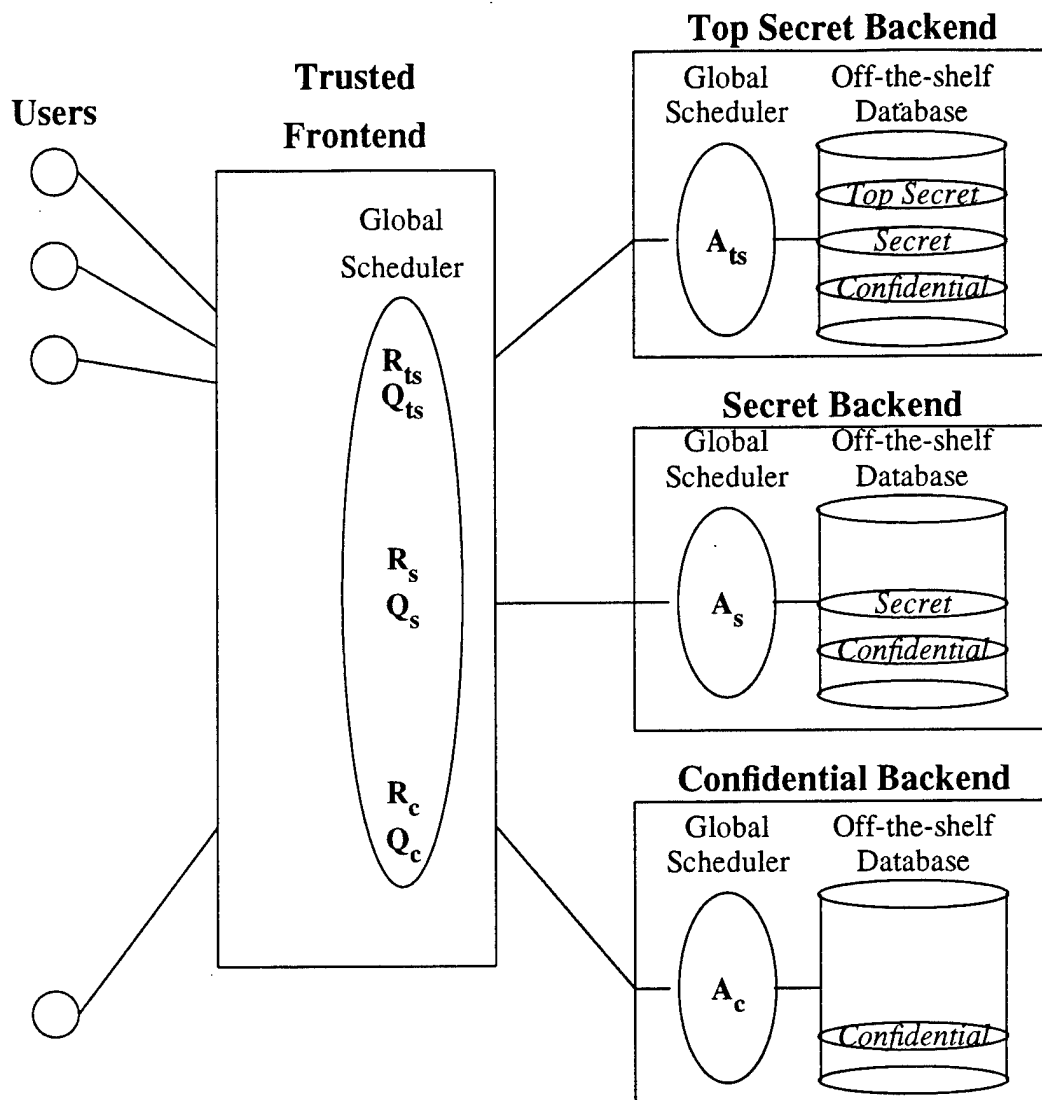


Figure 1: The SINTRA Architecture.

In the SINTRA project, we make the following assumptions:

- (1) All UBD use the same database query language (e.g., SQL).
- (2) The TFE changes the database states of the UBD only through database queries.
- (3) Each UBD performs some type of scheduling which produces a serializable and recoverable history.

1.1. Merits and Problems for the Replicated Architecture

At first glance, a database management system for each security level may seem excessive. However, we think this approach has the following merits:

- The security policy can be easily enforced by carefully designing interfaces among different database systems.
- Development cost can be reduced because commercial database systems for backend computers are widely available.
- The amount of trusted software can be minimized.
- Performance can be improved by using optimization and parallelization techniques which have been developed for conventional databases. This is the case because the replicated architecture uses conventional database systems as backend database systems, and uniprocessor or multiprocessor computers can be chosen as backend computers without affecting the security policy.

Despite the above advantages, the replicated architecture has a unique problem. Since each UBD in a replicated architecture contains data from lower levels, update transactions have to be propagated up to higher security level databases. There are some problems which are related to this propagation.

- [a] Since lower level update transactions propagate to higher level databases, high-level databases can be overloaded with lower level update transactions. This creates problems such as slow response to the request of a high-level user and longer propagation time for lower level update transactions.
- [b] The propagation of update transactions has to be carefully controlled. If the propagation of update transactions is not carefully controlled, inconsistent database states among backend databases can be created. Consider this example. Two confidential level update transactions T_i and T_j are serialized in the order of $\langle T_i, T_j \rangle$ at the confidential level backend database system. Since these two transactions are update transactions, these transactions have to be propagated to the secret level. If these two transactions are serialized in the order of $\langle T_j, T_i \rangle$ at the secret level, an inconsistent database state between confidential and secret level backend databases may be created. Therefore, the serialization order introduced by the local scheduler at the user's session level must be maintained at the higher

level UBDs.

A possible solution to problem [a] is presented in [10]. In this paper we concentrate on solutions to problem [b].

1.2. Motivation for Another Concurrency Algorithm

Several concurrency control mechanisms which preserve database consistency and security for the replicated architecture have been proposed [4, 8, 12]. Those concurrency control algorithms assume that each UBD uses conservative scheduling or something similar to preserve the scheduled order of conflicting updates (i.e., never abort update transactions from lower security classes). In reality, off-the-shelf database systems do not generally guarantee this condition. Also such scheduling may either pass the burden to the user by asking him to predeclare read and write sets or remove the interactive query capability of database system. Hence, this assumption poses performance and usability problems for the SINTRA project.

Also the proposed algorithms use the conventional basic operations, read and write, to describe transactions. Traditionally, $r[x]$ and $w[y]$ are used to denote "read data item x " and "write data item y ," respectively. Data items x and y may be relations, fixed-sized pages, or tuples depending on the granularity of concurrency controllers. In this paper, we propose a new transaction model which is better suited for the SINTRA architecture.

The scheduler for the SINTRA architecture has two kinds of components; global and local. The global scheduler enforces data consistency among the UBDs. On the other hand, the local scheduler enforces serializability among transactions which are submitted to the backend database system. In theory, where the global scheduler executes is not important in this architecture. However, since we expect that I/O will be a bottle-neck for this type of architecture, we distribute much of the single level part of the global scheduler to the backend computers, depicted in figure 1. The local schedulers are the concurrency controllers of the off-the-shelf database systems.

The local schedulers typically use locks or timestamps based on the knowledge of actual data or physical layout of the data in each UBD. However, the global scheduler has very little knowledge about the behavior of the local scheduler or the physical layout of data. For example, the global concurrency controller has no knowledge about where a specific tuple is located or which physical page should be locked. Sometimes the tuples which will be modified are unknown until the computation based on existing data is completed. The above factors may force the global concurrency controller to use relations as basic units to detect conflicts among transactions. Such a scheduler will be too restrictive and inefficient because it ignores the fact that referencing only a few tuples or few attributes of a relation is not the same as referencing the entire relation.

Based on the observations above, we argue that the traditional transaction model is not sufficient to model transactions for this replicated architecture. In this paper, we introduce a layered view of transactions. In our model, a transaction can be viewed as a sequence of database queries, and each query can be viewed as a sequence of read and write operations. Based on this model, we introduce a concurrency control algorithm which makes no assumption that each UBD uses any particular scheduling technique.

This paper is organized as follows. Section 2 discusses a transaction model for the SINTRA architecture. A concurrency control mechanism based on this transaction model is presented in section 3. Finally, section 4 summarizes the contributions of this paper.

2. THE MODEL

In this section, models are presented for security, replicated architecture, and transaction processing. The transaction model, which is presented in this section, can alleviate the difficulties described above in section 1.2.

2.1. Security Model

The security model used here is based on that of Bell and LaPadula [1]. The database system consists of a finite set D of *objects* (data item) and a set T of *subjects* (transactions). There is a lattice S of security classes with ordering relation $<$. A class S_i *dominates* a class S_j if $S_i \geq S_j$. There is a *labeling function* L which maps objects and subjects to a security class:

$$L: D \cup T \rightarrow S$$

Security class u *covers* v in a lattice if $u > v$ and there is no security class w for which $u > w > v$.

We consider two mandatory access control requirements:

(Simple Security Property)

If transaction T_i reads data item x then $L(T_i) \geq L(x)$.

(Restricted *-Property)

If transaction T_j writes data item x then $L(T_j) = L(x)$.

The simple security property allows a transaction to read data items if the security level of a transaction dominates the security level of data items. The restricted *-property allows a transaction to write if the security level of a transaction is the same as that of data items (i.e., no write-ups or write-downs are permitted). In [8], it is argued that write-ups (i.e., T_i cannot write to data item x if $L(T_i) < L(x)$) are undesirable in database systems for integrity reasons.

2.2. Replicated Architecture Model

The system has a TFE, which mediates the access of subjects to objects. The TFE contains the trusted computing base (TCB), but not all of the TFE need to be trusted. The system also contains a set of single level untrusted backend databases C , one for each element of the security lattice. Each backend database C_u contains copies of all data items in all databases whose security level is dominated by security level u .

Alternatively, if $L(x) = u$ such that $x \in C_u$, then there is a copy of x in each database whose security level dominates u .

2.3. Transaction Model

We adopt a layered model of transactions, where a transaction is a sequence of queries, and each query can be considered as a sequence of reads and writes. For example, replace and delete queries can be viewed as a read operation followed by a write operation, insert can be viewed as a write operation, and retrieve can be viewed as a read operation. A layered view of two transactions T_1 and T_2 is shown in figure 2.

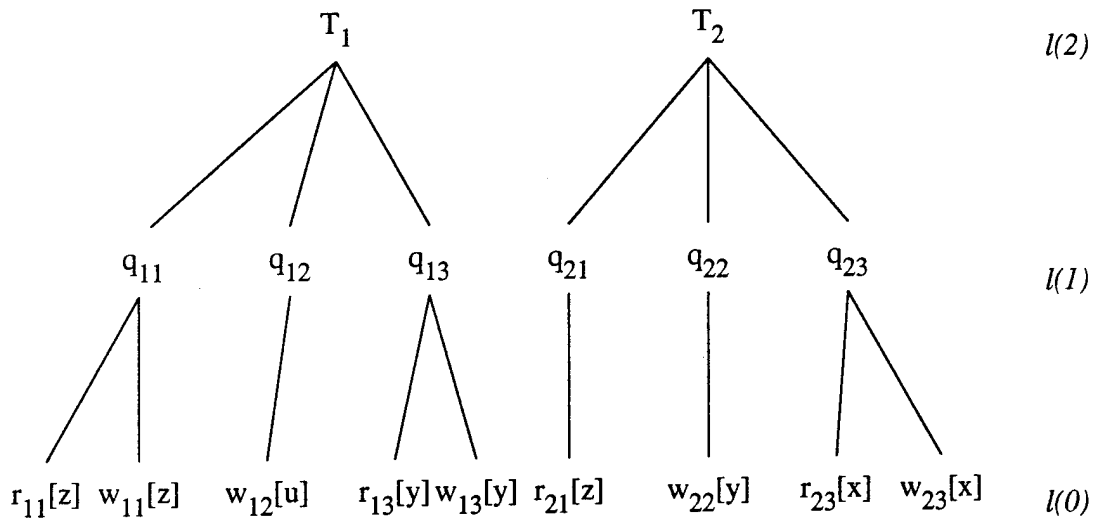


Figure 2: Layered model of two transactions.

Definition 1.

A transaction T_i is a sequence of queries, i.e., $T_i = \langle q_{i1}, q_{i2}, \dots, q_{in} \rangle$. Each query, q_{ij} , is an atomic operation and is one of retrieve, insert, replace, or delete.

To model the propagation of updates produced by a given transaction to higher security level databases, *update projection* is defined.

Definition 2.

An **update projection** U_i , which corresponds to a transaction T_i , is a sequence of update queries, e.g., $U_i = \langle q_{i2}, q_{i5}, \dots, q_{in} \rangle$ obtained from transaction T_i by simply removing all retrieve queries.

Note that our update projections consist of read and write operations.

To describe concurrency control mechanisms, we adopt the following definition of *conflict*.

Definition 3.

Two operations at the same layer **conflict** if and only if there is a possible state in which they do not commute. Alternatively, two operations conflict if they operate on common data and not both are retrieve operations.

It is interesting to compare our transaction model to another multilevel transaction model which appears in [13]. In their model, the low-level conflicts impose constraints on the serialization orders for higher levels. However, in the SINTRA architecture, the global scheduler does not have enough information about the conflicts which may occur at the local scheduler. Therefore, the global scheduler has to make serialization decisions independent of the those of the local scheduler.

In the following section, we present a concurrency control algorithm using the transaction model above. In our concurrency control algorithm, the global scheduler works at the query level (i.e., $l(1)$ in figure 2).

3. A CONCURRENCY CONTROL MECHANISM

In this section, a concurrency control algorithm is presented, which makes no assumption about UBD scheduling. In this algorithm, two types of schedulers can be identified, global and local schedulers. The global scheduler enforces data consistency among different security levels. On the other hand, the local scheduler enforces serializability among transactions, including update projections, which are submitted to the backend database system. The local scheduler deals with layer $l(0)$ in figure 2, and the global scheduler deals with layer $l(1)$ and upper layers. The global scheduler detects conflicts at level $l(1)$. Therefore, no knowledge of the specific items to be accessed or even the granularity of the lower level concurrency controller is required.

3.1. Algorithm C

To describe the concurrency control protocol, we need to define several mechanisms:

- A queue Q_u is associated with each backend database C_u , where u is a security level. The purpose of Q_u is to maintain a list of update projections which have been executed and committed at C_u . The queue is ordered by the serialization order of the execution of these transactions at C_u .
- In addition, there is an untrusted mechanism R_u which maintains Q_u and can read the contents of Q_v for all v which are dominated by u in the security lattice.
- Another queue A_u is associated with each backend database C_u . The purpose of A_u is to maintain a list of update projections which come from Q_v , where v is covered by u , and are waiting to be sent to C_u . The order of update projections in A_u is determined by the concurrency control algorithm which will be described later.

In our algorithm, Q_u , A_u , and R_u are considered parts of a global scheduler. Since mechanism R_u has to read the contents of Q_v for all v which are dominated by u , the R_u and the Q_u may be located in the TFE. However, A_u may be located in the

backend system (see figure 1). Also in our algorithm, we say a backend database C_u covers C_v if u covers v in the security lattice. The protocol processes transactions as follows:

Algorithm C:

At each backend database C_u :

- [1] Primary transactions (that are submitted directly by the user) and update projections are received from the global scheduler and submitted to the local backend scheduler.
- [2] As local transactions (primary transactions and update projections) are committed, a report of their serialization is sent to the global scheduler. These reports are sent in an order consistent with the serialization order determined by the local scheduler.

At the global scheduler:

- [1] For each primary transaction T_i submitted to the TFE, T_i is dispatched to C_u for processing where $L(T_i) = u$.
- [2] Whenever a serialization report for T_i or U_j is received from C_u , it is added to the end of Q_u .
- [3] The R_u scans the queue Q_v for those v for which C_u covers C_v . The R_u will retrieve an update projection U_i from Q_v and add it to the end of A_u when the following condition is satisfied for all $v \in S$:
 - If C_u covers C_v , and U_j can eventually appear in Q_v , then it does appear in the beginning of Q_v .
- [4] For update projections in the queue A_u , update projections are sent to C_u one after another. Specifically, if U_i is before U_j in the queue A_u , then send U_i and wait until U_i is committed at C_u , and then send U_j .
- [5] An update projection is removed from A_u once it is committed.
- [6] If an update projection, U_i , is aborted then resubmit U_i to C_u .

In algorithm C, we assume that local schedulers produce schedules such that the serialization order and the commit order of transactions are the same² (i.e., for any pair of transactions T_i and T_j , if T_i is committed before T_j then T_i also precedes T_j in the

serialization order). However, most database schedulers do not guarantee the above condition. The *take-a-ticket* [7] operation can be used to force any *recoverable* scheduler [2] to produce schedules such that the serialization order and the commit order of transactions are the same. The *take-a-ticket* operation consists of reading the value of a ticket prior to commit time, and incrementing it through regular data manipulation operations. The value of a ticket determines the serialization order. All operations on tickets are subject to the local concurrency control.

Note that algorithm C does not slow down user (primary) transactions. The global scheduler of algorithm C concerns the serialization order of the update projections in A_u at each security level. Concurrency control among primary transactions and update projections is the responsibility of the local scheduler in the UBD.

Also note that Q_u and R_u are not needed if the security classes form a completely ordered set, since A_u satisfies all the requirement of the algorithm.

3.2. Proof of Correctness

Many concurrency control algorithms have been proposed for the replicated architecture [4, 8, 12]. These papers use *one-copy-serializability* (or *ISR*) [2] as the correctness criteria for their concurrency control algorithms. In this paper, we present an alternative correctness criteria which we consider more intuitive. Based on this criteria, we will prove that algorithm C is correct. We will then show that our criteria imply one-copy-serializability.

For the sake of mathematical convenience, in this section, we assume read-only transactions have empty update projections which serve solely to mark their position in the serialization ordering of all transactions. Also, in this section, we do not

2. Consider the history of two transactions, T_i and T_j , H , where $H = r_i[x] w_i[x] r_j[x] w_j[x] w_i[y] c_i c_j$. This history does not satisfy the rigorousness condition [3]. However, this history will be satisfactory for our purpose because the serialization order and the commit order of transactions are the same.

distinguish the queue Q_u at the security class u and the contents of update projections in the queue Q_u .

An example will help to clarify our approach. Consider the security lattice in figure 3, and two non-conflicting **L**-level transactions T_i and T_j . Also consider an **M1**-level transaction T_u , and an **M2**-level transaction T_v . Let's further assume that T_u conflicts with T_i and T_j , and T_v conflicts with T_i and T_j . Since two transactions, T_i and T_j , are not conflicting and our security model does not allow *write-down*, an execution order $\langle T_i, T_u, T_j \rangle$ at security class **M1** and an execution order $\langle T_j, T_v, T_i \rangle$ at security class **M2** will generate the same result on *replicas* of security class **L** data. However, the reversed order between T_i and T_j at security classes **M1** and **M2** will create confusion in applying our update projection propagation rule. Specifically, at security class **H**, a consistent ordering among T_i , T_j , T_u , and T_v cannot be determined then *ISR* will be violated. Consequently, any global scheduler which does not enforce the same ordering among transactions at each relevant UBD may fail to produce consistent schedules. Thus any algorithm which gives *ISR* schedules must preserve the orderings at lower levels.

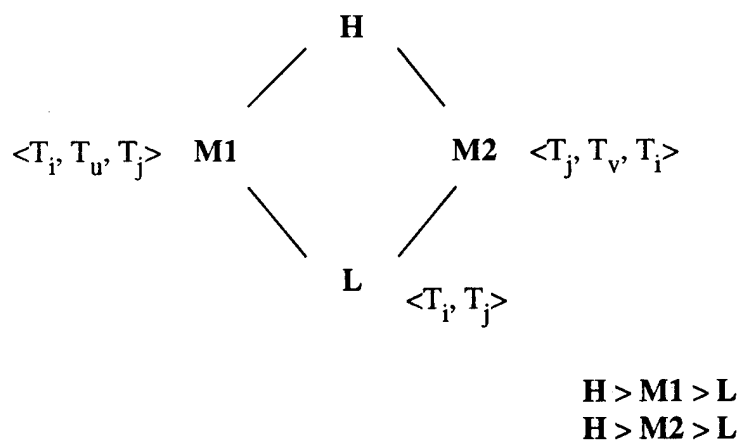


Figure 3: A security lattice

In this paper, we use another criteria which says “preserve the order between update projections which is determined at lower security class (or preserve the relative order).” We also show that any schedule for this architecture which preserves the relative order of update projections satisfies *ISR*.

Let $U = \{ U_i \mid T_i \in T \}$ be the set of all update projections and U^* be the set of all strings from U . Then for each pair of security classes u and v of S , for which $u \geq v$, there is a projection $\pi_{uv} : U^* \rightarrow U^*$ which is defined as follows:

- (a) $\pi_{uv}(U_i) = U_i$ if $L(T_i) \leq v$.
- (b) $\pi_{uv}(U_i) = \lambda$, the null string, otherwise.

Definition 4.

If u and v are in S , with $u \geq v$, and $\pi_{uv}(Q_u) = Q_v$ then we say that the **relative order** between Q_u and Q_v is preserved.

For example, if $Q_u = \langle U_i, U_k, U_j \rangle$ and $Q_v = \langle U_i, U_j \rangle$, and U_k is originated from security class w , where $u \geq w > v$, then the relative order of update projections in Q_u and Q_v is preserved because $\pi_{uv}(Q_u) = Q_v = \langle U_i, U_j \rangle$. We can now state our concept of correctness for transaction processing more precisely.

Definition 5.

For replicated architecture trusted database systems as above, let Q_u be the serialization order of the transactions and update projections committed at C_u . Then a concurrency control algorithm is **correct** if it preserves relative orders for all elements of the security lattice.

Theorem 1

Algorithm C is correct.

Proof.

This is evident from the algorithm, since at each class, the update projections are executed and committed in the same relative order established at lower security class. \square

Briefly, a schedule of transaction execution on a replicated database system is *one-copy-serializable* if it is *view equivalent* to a serial schedule on a one-copy database systems. View equivalence requires the two schedules to have the same reads-from and final-writes relationships. Details for this architecture may be found in [4].

Such schedules will be referred to as **ML-ISR**. Although in the following proofs, we use only read and write operations, we could instead use `retrieve`, `insert`, `replace`, and `delete` as in our query languages. We denote read and write operations on a data item x , or a copy of it x_n , of transaction T_i , by $r_i[x]$ and $w_i[x]$.

Theorem 2

With the architecture as specified above, if each UBD has a local scheduler which produces serializable schedules and there is a global scheduler such that for all u and v with $u \geq v$, the relative order between Q_u and Q_v is preserved, then the global schedule is **ML-ISR**.

Proof.

Let S_m , where m is the maximal element of the security lattice, determine a serial execution order on the one-copy logical database corresponding to the replicated system, and let H be the schedule produced by an algorithm satisfying the hypotheses of the theorem. We assert that H is view equivalent to S_m . Clearly, H and S_m have the same final-writes, by definition of S_m . We will show that H and S_m have the same reads-from relationships using proof by contradiction.

- (1) Suppose T_j reads- x -from T_i in S_m , but not in H . Let $L(T_j) = n$. Then in C_u , $w_i[x]$ precedes $w_k[x]$ and $w_k[x]$ precedes $r_j[x]$. But then at Q_u , the serialization order is T_i precedes T_k and T_k precedes T_j . This is preserved in S_m , by hypothesis, contradicting that T_j reads- x -from T_i in S_m .
- (2) Suppose T_j reads- x -from T_i in H , but not in S_m . Then if $L(T_j) = n$, T_j reads- x -from T_i in C_u . But if there is a T_k for which $w_i[x]$ precedes $w_k[x]$ and $w_k[x]$ precedes $r_j[x]$ in S_m , then because $L(T_k) = L(T_i) = L(x) \leq n$, this relationship also holds in C_u and thus in H . This contradicts our assumption that T_j reads- x -from T_i in H . \square

It is worthwhile to note that if the system has a completely ordered security lattice, the relative order between non-conflicting update projections does not have to be preserved. It is sufficient to preserve the ordering of conflicting update projections, rather than all update projections. This may permit greater concurrency.

Corollary 1

Algorithm C produces **ML-ISR** schedules.

Proof.

This follows immediately from the preceding theorems. \square

3.3. Two Variations

Step [4] of algorithm C forces update projections to execute serially. However, if the global scheduler of the SINTRA system can detect conflicts among transactions, we can achieve better concurrency among update projections by taking advantage of this knowledge. Since the backend database system usually cannot report whether there were conflicts, an accurate analysis technique which can detect conflicts among transactions is needed. Data dependence analysis, introduced in [9], can detect conflicts among transactions without any knowledge of actual data or physical layout of data. Rather, it relies on analysis of the queries themselves, detecting conflicts by determining if common data is to be accessed.

The rest of the section introduces two variations of the algorithm C, *optimistic* and *semi-optimistic* approaches, which may achieve better concurrency. These variations are concerned with how update projections in A_u are submitted to the UBD, and therefore how committed user transactions and update projections are placed in Q_u . In our two variations, transactions are executed hoping that the "correct" schedule is produced. When it is not, some amount of work already done will have to be undone. Hence, processing that is completed will have to be *certified* before it can be committed.

For the rest of the section, we assume that the serialization order is known at the end of each transaction (just before it might be committed). No transaction may actually be committed unless the global scheduler certifies it. If a user transaction or an update projection is committed, then it will be dispatched to Q_u . However, if a user transaction or an update projection is executed but not yet committed, the global

scheduler will store it in a queue P_u . P_u holds candidates for insertion into Q_u . Once the global scheduler certifies and commits a transaction then it will be moved from P_u to Q_u .

3.3.1. Optimistic Approach

Generally, the optimistic variation of the algorithm C works as follows. User transactions and update projections at any security level are submitted to the backend as they arrive. If an update projection is completed out of the submission order, it is held in P_u awaiting certification, along with the completed user transactions submitted at that level. When an update projection which is submitted earlier completes and is placed in P_u , data dependence analysis is used to determine whether the serialization order determined by P_u is equivalent (in a technical sense) to a correct (update order-preserving) one. If it is, P_u is rearranged to get a maximal prefix which is correct. The transactions in the prefix are certified and committed. If P_u cannot be rearranged, all transactions in P_u must be rolled back and re-submitted.

More specifically, the optimistic approach works as follows:

- [a] Submit update projections in A_u to UBD as they arrive.
- [b] Commit user transactions and update projections as long as update projections are serialized in the order that they are submitted to UBD up to that time (i.e., P_u is empty). Once those are committed then dispatch them to Q_u .
- [c] If U_i should be next to be serialized but U_j is already serialized then put U_i in P_u without committing it. Any user transactions or update projections which are executed will be put in P_u without committing them until U_i is executed. After U_i is executed, put U_i into P_u .
- [d] While the first and last transactions in P_u are update projections, do the following steps:
 - (1) Find an update projection in P_u which should be serialized before any other uncommitted update projections. Call that update projection T_n .

- (2) Test if the first transaction in P_u , T_1 (which must be an update projection), conflicts with T_n . If T_1 and T_n conflict then abort and remove all transactions in P_u , re-submit them, and return to [a].
- (3) Test if T_1 conflicts with any of the transactions from T_2 through T_{n-1} . If there is no conflict, then move T_1 to immediately after T_n in P_u and go to step (5).
- (4) Test if T_n conflicts with any of the transactions from T_2 through T_{n-1} . If there is a conflict, then abort and remove all transactions in P_u , re-submit them, and return to [a]. Otherwise move T_n to the beginning of P_u .
- (5) While the first transaction in P_u , T_1 , is either a user transaction or an update projection which is in the proper order to be serialized then do the following steps:
 - Commit T_1 , remove it from P_u and A_u if applicable, and dispatch it to Q_u .

An example of this may be helpful. Consider a situation where the global scheduler submits update projections in the order of $\langle U_i, U_j \rangle$ and the serialization order at the UBD is $\langle U_j, T_k, U_i \rangle$ where T_k is a user transaction. Since U_j is serialized before U_i , U_j cannot be committed until U_i is committed. Now the task is to find if the order of either U_i or U_j can be rearranged. The only way this can happen is:

- There is no conflict between U_i and U_j , and either
 - (a) There is no conflict between U_j and T_k , or
 - (b) There is no conflict between T_k and U_i .

If situation (a) happens then $\langle T_k, U_i, U_j \rangle$ will be the order which will be sent to Q_u by the algorithm. If situation (b) happens then $\langle U_i, U_j, T_k \rangle$ will be the order which will be sent to Q_u . This more complex approach may be justified if conflicts are likely to occur more frequently.

We could improve the performance of this algorithm by minimizing the number of transactions which must be aborted. For example, assume that T_1 and T_n do not conflict but T_1 conflicts with T_k and T_n conflicts with T_{k+1} . We could just abort T_{k+1} through T_n to ensure that T_n is serialized before T_{k+1} .

3.3.2. Semi-optimistic Approach

Since the SINTRA security policy prohibits *write-up* or *write-down*, the probability of conflict between a user transaction at security class u and an update projection from the security class v where $u > v$ may be quite small³. Hence if conflicts among update projections are detected before those are submitted then there is less probability of aborting.

The semi-optimistic variations is similar to the optimistic one, but rather than submitting update projections as they arrive, it checks for conflicts first, thereby reducing the likelihood of having to roll back work already done. Specifically, the semi-optimistic approach replace the step [a] of the preceding optimistic approach with following:

- [a1] Detect conflicts among update projections in A_u .
- [a2] If two update projections U_i and U_j conflict then submit them serially (i.e., submit one and then wait for a commit message before submitting another).
- [a3] If there are update projections in A_u which do not conflict then submit one after another (i.e., submit one and then submit next update projection without waiting for commit message of previous update projection).

After applying steps [b] and [c] of the optimistic approach, P_u can be tested, as before (i.e., step [d] of optimistic approach). The only difference is that there is no need to detect conflicts among update projections because those have been already tested. Therefore, only steps (1), (3), (4), and (5) from the optimistic approach are required. This technique clearly falls between those of the previous two algorithms. The following table clarifies the different approaches of the three variations.

3. The SINTRA security policy allows *read-down*. Hence, a user transaction at level u can conflict with an update projection from level v where $u > v$.

Algorithm Variant	Submission Process for Update Projections	Mechanism for Insuring Consistent Update Projection Ordering
Pessimistic (Algorithm C)	One at a time	None required
Semi-optimistic	Check for conflicts before submitting. Maintain submission order for conflicting update projections.	Check for conflicts between update projections and primary transactions after execution. Roll back and redo as necessary.
Optimistic	As they arrive (No checking or delaying)	Check for conflicts among all local transactions after execution. Roll back and redo as necessary.

3.3.3. Correctness of the Variations

Corollary 2

The Optimistic and semi-optimistic variations of algorithm C produce **ML-1SR** schedules.

Proof.

This is evident from the algorithm, since the global scheduler certifies a schedule only if the relative order between Q_u and Q_v is preserved for all u and v with $u \geq v$. \square

4. CONCLUSIONS

In this paper, we have presented arguments that the traditional transaction model is not sufficient to model transactions for multilevel-secure databases which use the replicated architecture. We have proposed a new transaction model for multilevel-secure databases.

Even though several concurrency control algorithms for the replicated architecture have been proposed, those algorithms assume that each UBD uses conservative scheduling or at least assumes schedules preserve the order of update projections without indicating how it is done. We present a concurrency control algorithm which does not assume that each UBD uses conservative scheduling. Our concurrency control algorithm is based on the transaction processing model which is proposed in this paper, which controls ordering through other means, outside the UBD.

Our basic concurrency algorithm, algorithm C, executes update projections serially. We also offer two variations of algorithm C, optimistic and semi-optimistic approaches, which may achieve better concurrency under a variety of likely conditions. Our directions for future research are performance comparison among different variations under different application scenario. These algorithms are, in fact, being implemented in our prototype for the SINTRA project.

References

- [1] Bell, D. E., and LaPadula, L. J. Secure computer systems: Unified exposition and multics interpretation. The Mitre Corp, (1976).
- [2] Bernstein, P. A., et al. Concurrency controller and recovery in database systems. Addison-Wesley (1987).
- [3] Breitbart, Y., et al. On rigorous transaction scheduling. IEEE Transaction on Software Engineering, 17, 6 (1991).

- [4] Costich, O. Transaction processing using an untrusted scheduler in a multilevel database with replicated architecture. in Database Security V: Status and Prospects (North-Holland 1992)
- [5] Denning, D. Commutative filters for reducing inference threats in multilevel database systems. Proceedings of the IEEE symposium on Security and Privacy (1985).
- [6] Froscher, J. N., and Meadows, C. Achieving a trusted database management systems using parallelism. in Database Security II: Status and Prospects (North-Holland 1989)
- [7] Georgakopoulos, D., et al. On serializability of multidatabase transactions through forced local conflicts. Proceedings of Conference on Data Engineering (1991).
- [8] Jajodia, S., and Kogan, B. Transaction processing in multilevel-secure databases using replicated architecture. Proceedings of the IEEE symposium on Security and Privacy (1990).
- [9] Kang, M. H., et al. Data dependence analysis for an untrusted transaction manager in a multilevel database system. Submitted for publication (1992).
- [10] Kang, M. H. Design criteria and methods for a multilevel database system. In preparations.
- [11] Lunt, T., et al. The seaview security model. IEEE Transaction on Software Engineering, 16, 6 (1990).
- [12] McDermott, J., et al. A single level scheduler for the replicated architecture for multilevel-secure databases. Proceedings of the seventh annual computer security applications conference (1991).
- [13] Weikum, G. Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, 16, 1 (1991)

Tuple-level vs element-level classification*

Xiaolei Qian and Teresa Lunt

Computer Science Laboratory, SRI International
333 Ravenswood Avenue, Menlo Park, CA 94025

Abstract

A multilevel relational database represents information in a multilevel state of the world, which is the knowledge of the truth value of a statement with respect to a security level. The security semantics of a data classification scheme specifies the way that the classification of data in a multilevel relational database corresponds to the classification of statements in a multilevel state of the world. We formalize the security semantics of tuple-level and element-level data classification schemes, and show that they have the same expressive power. We derive entity, referential, and polyinstantiation integrity properties from the security semantics, and show that existing approaches to polyinstantiation integrity are inappropriate. Our results provide the foundation for the multilevel relational model, its integrity properties, its operational semantics, and its implementation.

1 INTRODUCTION

A *state of the world* can be envisioned as a set of elements linked together by relationships and functions. Information in a state of the world is the knowledge of the truth value of a statement[9], which can be either an elementary fact as "Enterprise is on a spy mission to Rigel" or a general law as "every ship has a unique destination".

A *relational database* captures a finite set of elements linked together by relationships and functions. Relationships are represented as relations, and functions are represented as functional and referential dependencies. Every tuple in a relation represents the truth of an elementary fact, and every functional or referential dependency represents the truth of a general law. These are the only information explicitly captured by a relational database, from which implicit information can be derived. For example, from the explicit elementary fact "Enterprise is on a spy mission to Rigel" represented by the tuple "(Enterprise, spy, Rigel)" in relation Starship-Mission-Destination, we can derive the implicit information "there is a ship Enterprise".

A *multilevel state of the world* is a state of the world together with a *classification mapping*: every piece of information — either an elementary fact or a general law —

*This work was supported by U.S. Department of Defense Advanced Research Projects Agency and U.S. Air Force Rome Laboratory under contract F30602-91-C-0092 for Inference-Channel Detection and Elimination in Knowledge-Based Systems.

is mapped to a *security level* in a *classification lattice*. Information in a multilevel state of the world is the knowledge of the truth value of a statement with respect to a security level[12], which can be either a classified elementary fact as “it is top-secret that Enterprise is on a spy mission to Rigel”, or a classified general law as “it is confidential that every ship has a unique destination”.

A *multilevel relational database* is a relational database together with a *data classification scheme*, which is a mapping of every data item in the relational database to a security level in a classification lattice. We consider two data classification schemes: *tuple-level classification* and *element-level classification*¹. Tuple-level classification considers every tuple in every relation to be a data item, while element-level classification considers every element of every tuple in every relation to be a data item.

The data classification scheme of a multilevel relational database is intended to represent the classification mapping of elementary facts of a multilevel state of the world. More specifically, for every elementary fact in the multilevel state of the world that is captured by some data item in the multilevel relational database, the classification of the elementary fact under the classification mapping should be equal to the classification of the data item under the data classification scheme. Moreover, the classification of every data item in the multilevel relational database under the data classification scheme should represent the classification of some elementary fact in the multilevel state of the world under the classification mapping. In other words, the diagram in Figure 1 should commute. This correspondence between the data classification scheme of a multilevel relational database and the classification mapping of elementary facts of a multilevel state of the world establishes the *security semantics* of the data classification scheme.

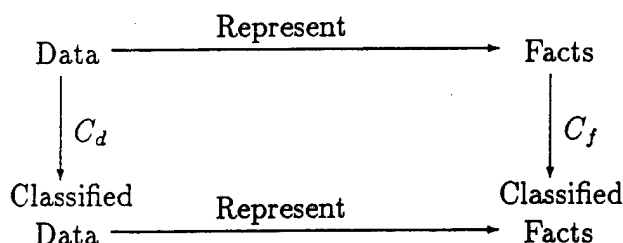


Figure 1: Data vs Fact Classification

The security semantics of tuple-level classification can be easily specified. Because every tuple in a relation represents the truth of an elementary fact in a state of the world, the classification of the tuple under tuple-level classification naturally represents the classification of the elementary fact under the classification mapping. For example, if tuple “(Enterprise, spy, Rigel)” is classified top-secret under tuple-level classification, then the elementary fact “Enterprise is on a spy mission to Rigel” is classified top-secret under the classification mapping, and vice versa.

The security semantics of element-level classification is more problematic however.

¹Our results can be easily generalized to multilevel relational databases where a combination of data classification schemes is used, such as SeaView which uses both tuple-level and element-level classification.

Since elements in tuples of a relational database do not have direct correspondence to elementary facts in a state of the world, it is unclear what the correspondence is between the classification of an element in a tuple and the classification of any elementary fact. For example, element "spy" in tuple "(Enterprise, spy, Rigel)" could represent any of the elementary facts "Enterprise is on a spy mission", or "there is a ship whose mission is spy", or even "there is a ship on spy mission to Rigel". Hence the classification of the "spy" element in this tuple under element-level classification could represent the classification of any of these elementary facts under the classification mapping. It is therefore no coincidence that Thuraisingham chose tuple-level classification in her logical formalization of multilevel relational databases[12]. Neither is it accidental that the Franconia model of [11] imposes the requirement of *one tuple per tuple class*. We believe that a security semantics for element-level classification should be the foundation of any formal semantics of the multilevel relational model based on element-level classification.

Polyinstantiation integrity has been a subject of continuous debate in the literature[1, 3, 5, 8, 11]. Although polyinstantiation has been recognized as inevitable in a multilevel state of the world[2], no semantic justification has been given. Existing proposals are all based on informal arguments to eliminate certain intuitively undesirable relations. A restricted polyinstantiation integrity is proposed in [10], which not only introduces signaling channels but also causes loss of low information or denial-of-service to low users. It is suggested in [6] that whether a relation is desirable should be application dependent. While integrity constraints in the standard relational model are intended to capture general laws in a state of the world, it is unclear what general laws in a multilevel state of the world are captured by various notions of polyinstantiation integrity.

There have been several proposals on an operational (update) semantics for the multilevel relational model[1, 4, 6, 7, 8], all of which impose integrity properties such as entity integrity and various notions of polyinstantiation integrity. Since these integrity properties are justified only informally[8, 11] (often based on intuition or implementation considerations), the operational semantics that derives from them is unavoidably controversial. It is worth noticing that the semantics of the standard relational model — either model-theoretic or proof-theoretic — is defined completely independent of its operational semantics[9]. We think that a lot of the controversy surrounding the operational semantics of the multilevel relational model is due to the lack of a formal basis of polyinstantiation integrity, and this formal basis should be formulated based on the security semantics of element-level classification, rather than the operational semantics.

Intuitively, element-level classification seems to be more expressive than tuple-level classification, because it is more fine-grained in capturing the classification of elementary facts in a multilevel state of the world. On the other hand, element-level classification seems to be more complicated and difficult to implement than tuple-level classification. A formal characterization of the expressive power of these data classification schemes would be invaluable in making design decisions such as which scheme to choose in building a multilevel relational database.

We formalize the security semantics of **element-level classification** in multilevel relational databases. Based on the security semantics, we show that element-level classification is equivalent to tuple-level classification in expressive power. As an application of the security semantics, we also show that the various models of polyinstantiation in-

egrity proposed in [11] are inappropriate. The paper is organized as follows. Section 2 formally defines the standard relational model. We characterize the information captured by the standard relational model in Section 3. Section 4 defines the security semantics of tuple-level and element-level classification, and shows that these two data classification schemes are equivalent in expressive power. We analyze in Section 5 the various models of polyinstantiation integrity proposed in [11], and show that they are inappropriate. Finally, Section 6 gives some concluding remarks.

2 RELATIONAL MODEL

The relational model can be defined formally as follows. Suppose U is a finite set of *attributes*. If X, Y are subsets of U , then XY denotes the union of X, Y . A *relation scheme* $R[X, K]$ is a set of attributes $X \subseteq U$ named R with non-empty *primary key* $K \subseteq X$. A *schema* $\mathcal{B} = (\mathcal{R}, \mathcal{C})$ is a set of relation schemes \mathcal{R} together with a set of *well-formed referential dependencies* \mathcal{C} :

- 1.1 every referential dependency in \mathcal{C} has the form $R_i[Y] \hookrightarrow R_j$, where $R_i[X_i, K_i]$ and $R_j[X_j, K_j]$ are relation schemes in \mathcal{R} , $Y = K_i$ or $Y \subseteq X_i - K_i$, and $|Y| = |K_j|$; and
- 1.2 $Y = Z$ or $Y \cap Z = \emptyset$ for R_i, R_j, R_k in \mathcal{R} and $R_i[Y] \hookrightarrow R_j, R_i[Z] \hookrightarrow R_k$ in \mathcal{C} .

The picture in Figure 2 shows a sample database schema, where boxes represent relation schemes, attributes on the left of double lines form primary keys, and arrows between boxes represent referential dependencies.

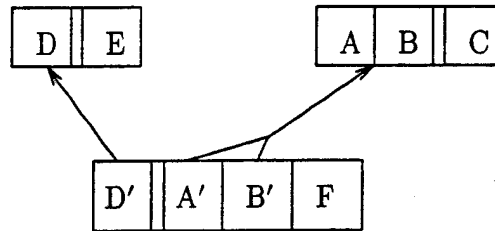


Figure 2: A Schema

Let \mathcal{D} be a (possibly infinite) set of values. A *tuple* over attributes X is a partial mapping $t[X]: X \mapsto \mathcal{D}$ that assigns values from \mathcal{D} to attributes in X . For $A \in X$, $t[A]$ denotes the value assigned to A by $t[X]$, and $t[A] = \perp$ denotes that $t[A]$ is undefined². For $Y \subseteq X$, $t[Y]$ denotes the partial mapping whose domain is restricted to attributes in Y , and $t[Y] = \perp$ ($t[Y] \neq \perp$) denotes that $t[A] = \perp$ ($t[A] \neq \perp$) for all $A \in Y$. A *relation* r over relation scheme $R[X, K]$ is a set of tuples over X such that the *primary key integrity property* holds:

²We distinguish between *unknown* nulls and *not-applicable* nulls. The symbol \perp represents unknown nulls. Unknown nulls say that some elementary facts are missing from the database. Because a database is not a part of the state of the world that the database tries to capture, unknown nulls do not represent elementary facts in the state of the world.

2.1 for every $t \in r$, $t[K] \neq \perp$, and

2.2 for every $t_1, t_2 \in r$, $t_1[K] = t_2[K]$ implies $t_1[X] = t_2[X]$.

In other words, tuples could be uniquely identified by their values in the primary key attributes. For $Y \subseteq X$, $r[Y]$ denotes the set of tuples $t[Y]$ where $t \in r$. A database b over schema \mathcal{B} with relation schemes $R_1[X_1, K_1], \dots, R_n[X_n, K_n]$ is a set of relations r_1, \dots, r_n where r_l is a relation over R_l for $l = 1, \dots, n$, such that the *foreign key integrity property* holds:

3.1 for every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} and $t_i \in r_i$, either $t_i[Y] = \perp$ or $t_i[Y] \neq \perp$, and

3.2 for every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} and $t_i \in r_i$ where $t_i[Y] \neq \perp$, there exists $t_j \in r_j$ such that $t_i[Y] = t_j[K_j]$.

In other words, the foreign key value of every tuple, if non-null, should refer to an existing tuple in the referenced relation. \mathcal{D} is the *universe* of b .

For $Y \subseteq X$, the *total projection* of relation $r[X]$ to Y , denoted as $\Pi_Y r[X]$, is defined as the set of tuples $t[Y]$ such that $t[Y] \in r[Y]$ and $t[Y] \neq \perp$. the *null extension* of relation $r[Y]$ to X , denoted as $r[Y] \uparrow X$, is defined as the set of tuples $t[X]$ such that $t[Y] \in r$ and $t[X - Y] = \perp$. For two relations $r_1[X_1]$ and $r_2[X_2]$, the *natural-join* $r_1 \bowtie r_2$ denotes the set of tuples $t[X_1 X_2]$ such that $t[X_1] \in r_1[X_1]$ and $t[X_2] \in r_2[X_2]$. The *outer-join* $r_1 \bowtie\!\!\!\bowtie r_2$ is defined as:

$$(r_1 - (r_1 \bowtie r_2)[X_1]) \uparrow X_1 X_2 \cup (r_1 \bowtie r_2) \cup (r_2 - (r_1 \bowtie r_2)[X_2]) \uparrow X_1 X_2$$

3 ATOMIC DECOMPOSITION

Every tuple in a database represents an elementary fact. Often, the elementary fact represented by a tuple is a conjunction of several *smaller* elementary facts. For example, tuple "(Enterprise, spy, Rigel)" represents the elementary fact "Enterprise is on a spy mission to Rigel", which is the conjunction of two smaller elementary facts "Enterprise is on a spy mission" and "Enterprise goes to Rigel". In this section, we show how to decompose a tuple into a set of smaller tuples, which represents a set of *smallest* elementary facts whose conjunction is equivalent to the elementary fact represented by the original tuple.

Suppose that $\mathcal{B} = (\mathcal{R}, \mathcal{C})$ is a schema, the *atomic decomposition* of \mathcal{B} is a schema consisting of the following relation schemes:

- $R^K[K, K]$ for every $R[X, K]$ in \mathcal{R} ,
- $R^Y[KY, K]$ for every $R[Y] \hookrightarrow R'$ in \mathcal{C} where $Y \subseteq X - K$, and
- $R^A[KA, K]$ for every $A \in X - K$ where $A \notin Y$ for any $R[Y] \hookrightarrow R'$ in \mathcal{C} ;

and the following well-formed referential dependencies:

- $R^Y[K] \hookrightarrow R^K$ and $R^A[K] \hookrightarrow R^K$ for every $R^K[K, K], R^Y[KY, K], R^A[KA, K]$,

- $R_i^K[K_i] \hookrightarrow R_j^K$ for every $R_i[K_i] \hookrightarrow R_j$ in \mathcal{C} , and
- $R_i^Y[Y] \hookrightarrow R_j^K$ for every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} where $Y \subseteq X_i - K_i$.

The picture in Figure 3 is the atomic decomposition of the sample database schema of Figure 2.

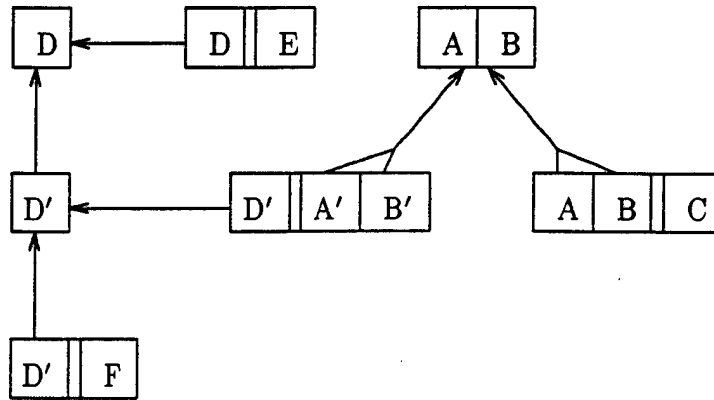


Figure 3: Atomic Decomposition

From every database b over $\mathcal{B} = (\mathcal{R}, \mathcal{C})$ we can construct a unique database $\delta(b)$ over the atomic decomposition $\mathcal{B}^a = (\mathcal{R}^a, \mathcal{C}^a)$ of \mathcal{B} as follows. From every relation $r \in b$ over $R[X, K]$ in \mathcal{R} , we construct relations $r^K = \Pi_{K^r} r$, $r^Y = \Pi_{KY^r} r$, and $r^A = \Pi_{KA^r} r$ in $\delta(b)$ over R^K , R^Y , and R^A respectively.

Theorem 1 *For every database b over $\mathcal{B} = (\mathcal{R}, \mathcal{C})$, $\delta(b)$ is a database over the atomic decomposition $\mathcal{B}^a = (\mathcal{R}^a, \mathcal{C}^a)$ of \mathcal{B} .*

Proof Obviously relations in $\delta(b)$ satisfy the primary key integrity property and the first requirement of the foreign key integrity property. Now we show that relations in $\delta(b)$ also satisfy the second requirement of the foreign key integrity property.

For every $R[X, K]$ in \mathcal{R} and $t \in r^Y$ in $\delta(b)$, there exists a $t' \in r$ in b such that $t = t'[KY]$ and hence $t'[K] \in r^K$. This means that $\delta(b)$ satisfies the referential dependency $R^Y[K] \hookrightarrow R^K$. Likewise we can show that $\delta(b)$ satisfies the referential dependency $R^A[K] \hookrightarrow R^K$. For every $R_i[K_i] \hookrightarrow R_j$ in \mathcal{C} and $t \in r_i^K$ in $\delta(b)$, there exists a $t_i \in r_i$ in b such that $t = t_i[K_i]$. Because b satisfies the referential dependency $R_i[K_i] \hookrightarrow R_j$, there exists a $t_j \in r_j$ in b such that $t_i[K_i] = t_j[K_j]$. But $t_j[K_j] \in r_j^K$ in $\delta(b)$, and hence $t \in r_j^K$. This means that $\delta(b)$ satisfies the referential dependency $R_i^K[K_i] \hookrightarrow R_j^K$.

For every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} and $t \in r_i^Y$ in $\delta(b)$, there exists a $t_i \in r_i$ in b such that $t = t_i[KY]$ and $t_i[Y] \neq \perp$. Because b satisfies the referential dependency $R_i[Y] \hookrightarrow R_j$, there exists a $t_j \in r_j$ in b such that $t_i[Y] = t_j[K_j]$. But $t_j[K_j] \in r_j^K$ in $\delta(b)$, and hence $t[Y] \in r_j^K$. This means that $\delta(b)$ satisfies the referential dependency $R_i^Y[Y] \hookrightarrow R_j^K$. \diamond

Similarly, from every database b^a over the atomic decomposition of \mathcal{B} , we can construct a unique database $\sigma(b^a)$ over \mathcal{B} as follows. Every relation $r \in \sigma(b^a)$ is constructed as the outer-join of relation r^K with all relations r^Y and r^A : $r^K \bowtie_Y r^Y \bowtie_A r^A$.

Theorem 2 For every database b^a over the atomic decomposition $\mathcal{B}^a = (\mathcal{R}^a, \mathcal{C}^a)$ of $\mathcal{B} = (\mathcal{R}, \mathcal{C})$, $\sigma(b^a)$ is a database over \mathcal{B} .

Proof Obviously relations in $\sigma(b^a)$ satisfy the primary key integrity property and the first requirement of the foreign key integrity property. Now we show that relations in $\sigma(b^a)$ also satisfy the second requirement of the foreign key integrity property.

For every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} and $t_i \in r_i$ in $\sigma(b^a)$ where $Y \subseteq X_i - K_i$ and $t_i[Y] \neq \perp$, we have that $t_i[KY] \in r_i^Y$ in b^a . Since b^a satisfies the referential dependency $R_i^Y[Y] \hookrightarrow R_j^K$, $t_i[Y] \in r_j^K$ in b^a . According to the definition of outer-join, there exists a $t_j \in r_j$ in $\sigma(b^a)$ such that $t_i[Y] = t_j[K_j]$. This means that $\sigma(b^a)$ satisfies the referential dependency $R_i[Y] \hookrightarrow R_j$.

For every $R_i[K_i] \hookrightarrow R_j$ in \mathcal{C} and $t_i \in r_i$ in $\sigma(b^a)$, we have that $t_i[K_i] \in r_i^K$ in b^a , or $t_i[K_i] \in r_i^Y[K_i]$ in b^a for some $Y \subseteq X_i - K_i$, or $t_i[K_i] \in r_i^A[K_i]$ in b^a for some $A \in X_i - K_i$. Suppose that $t_i[K_i] \in r_i^Y[K_i]$. Since b^a satisfies the referential dependency $R_i^Y[K_i] \hookrightarrow R_j^K$, $t_i[K_i] \in r_j^K$. Likewise, if $t_i[K_i] \in r_i^A[K_i]$ then $t_i[K_i] \in r_j^K$. Because b^a satisfies the referential dependency $R_i^K[K_i] \hookrightarrow R_j^K$, $t_i[K_i] \in r_j^K$ in b^a . According to the definition of outer-join, there exists a $t_j \in r_j$ in $\sigma(b^a)$ such that $t_i[K_i] = t_j[K_j]$. This means that $\sigma(b^a)$ satisfies the referential dependency $R_i[K_i] \hookrightarrow R_j$. \diamond

The existence of these unique constructions from b to b^a and vice versa implies that \mathcal{B} and its atomic decomposition \mathcal{B}^a capture exactly the same elementary facts, and hence are semantically equivalent. Notice that every tuple in b is in general broken into several tuples in b^a , every elementary fact captured by \mathcal{B} is therefore equivalent to a conjunction of several *smaller* elementary facts captured by \mathcal{B}^a . It is easy to verify that δ and σ are inverse mappings of each other:

Corollary 3 For every database b over \mathcal{B} , $\sigma(\delta(b)) = b$. For every database b^a over \mathcal{B}^a , $\delta(\sigma(b^a)) = b^a$.

Notice that the atomic decomposition of a database does not contain \perp (by the definition of the total projection operator Π). This implies that null values in a database do not represent elementary facts in a state of the world, which coincides with our intuition.

Furthermore, the atomic decomposition of \mathcal{B} into \mathcal{B}^a is the *finest* possible decomposition, in the sense that every tuple in \mathcal{B}^a represents an atomic elementary fact whose further decomposition leads to loss of information. For example, tuple "(Enterprise, spy)" represents the elementary fact "Enterprise is on a spy mission", while tuples "(Enterprise)" and "(spy)" represent the elementary facts "there is a ship Enterprise" and "there is a ship on a spy mission" respectively. The conjunction of the latter two is not equivalent to the former.

4 SECURITY SEMANTICS

A *multilevel schema* is a pair $(\mathcal{B}, \mathcal{L})$, where $\mathcal{B} = (\mathcal{R}, \mathcal{C})$ is a schema and $\mathcal{L} = (\mathcal{N}, \preceq)$ is a classification lattice consisting of a set of nodes \mathcal{N} and a partial order \preceq on \mathcal{N} . A

multilevel database over $(\mathcal{B}, \mathcal{L})$ is a pair (b, κ) , where b is a database over \mathcal{B} and κ is a *data classification scheme*. Figure 4 shows a classification lattice that we use in the rest of this paper.

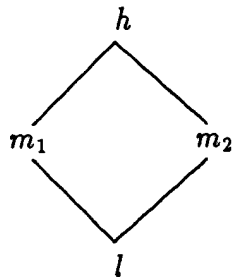


Figure 4: Classification Lattice

4.1 Tuple-level classification

For multilevel database (b, κ_t) with tuple-level classification, κ_t is a mapping where $\kappa_t(r, t) \in \mathcal{N}$ is a node in the classification lattice for relation $r \in b$ and tuple $t \in r$. We define the security semantics of tuple-level classification as follows. For every relation $r \in b$ and tuple $t \in r$, $\kappa_t(r, t)$ represents the classification of the elementary fact represented by t under the classification mapping of a multilevel state of the world.

Notice that the data items classified at the same level should satisfy all the functional dependencies. This requirement justifies the *polyinstantiation property* of tuple-level classification:

- 4 for every r over $R[X, K]$ in \mathcal{R} and $t, t' \in r$ where $t[K] = t'[K]$ and $\kappa_t(r, t) = \kappa_t(r, t')$, we have that $t = t'$.

A referential dependency $R_i[Y] \hookrightarrow R_j$ represents a function from every elementary fact represented by a tuple in r_i to some elementary fact represented by a tuple in r_j , and $r_i[Y]$ encodes this function in b . Notice that knowing a function between two elementary facts t_i, t_j demands knowing the two elementary facts first. This requirement justifies the *foreign key security property* of tuple-level classification:

- 5 for every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} , $t_i \in r_i$, and $t_j \in r_j$ where $t_i[Y] = t_j[K_j]$, we have that $\kappa_t(r_j, t_j) \preceq \kappa_t(r_i, t_i)$.

Figure 5 shows a multilevel database with tuple-level classification, over the schema of Figure 3. The classification of every tuple is specified to the right of the tuple.

4.2 Element-level classification

For multilevel database (b, κ_e) with element-level classification, κ_e is a mapping such that $\kappa_e(r, t, A) \in \mathcal{N}$ is a node in the classification lattice for relation r over $R[X, K]$ in \mathcal{R} , tuple $t \in r$, and attribute $A \in X$ where $t[A] \neq \perp$. As the result, null values in b are not

<table><tr><td>A</td><td>B</td></tr><tr><td>a</td><td>b</td></tr></table> l	A	B	a	b	<table><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>a</td><td>b</td><td>c</td></tr></table> h	A	B	C	a	b	c	<table><tr><td>D</td></tr><tr><td>d_1</td></tr><tr><td>d_2</td></tr></table> l	D	d_1	d_2	<table><tr><td>D</td><td>E</td></tr><tr><td>d_1</td><td>e_1</td></tr><tr><td>d_2</td><td>e_2</td></tr></table> m_1 m_2	D	E	d_1	e_1	d_2	e_2
A	B																					
a	b																					
A	B	C																				
a	b	c																				
D																						
d_1																						
d_2																						
D	E																					
d_1	e_1																					
d_2	e_2																					
<table><tr><td>D'</td></tr><tr><td>d_1</td></tr><tr><td>d_2</td></tr></table> m_1 m_2	D'	d_1	d_2	<table><tr><td>D'</td><td>A'</td><td>B'</td></tr><tr><td>d_1</td><td>a</td><td>b</td></tr></table> m_1	D'	A'	B'	d_1	a	b	<table><tr><td>D'</td><td>F</td></tr><tr><td>d_1</td><td>f_1</td></tr><tr><td>d_2</td><td>f_2</td></tr></table> h h	D'	F	d_1	f_1	d_2	f_2					
D'																						
d_1																						
d_2																						
D'	A'	B'																				
d_1	a	b																				
D'	F																					
d_1	f_1																					
d_2	f_2																					

Figure 5: Tuple-Level Classification

classified, which is natural because they do not represent elementary facts in the state of the world captured by b .

In order to determine a proper security semantics of element-level classification and the necessary security properties, let us consider the atomic decomposition of b into b^a and a multilevel database (b^a, κ_t) with tuple-level classification. Notice that both b and b^a capture exactly the same elementary facts, and every elementary fact represented in b is a conjunction of several elementary facts represented in b^a . Hence both κ_e and κ_t should capture exactly the same classification mapping of elementary facts in a multilevel state of the world. Hence we define the security semantics of element-level classification as follows. For relation r over $R[X, K]$ in \mathcal{R} , tuple $t \in r$, and attribute $A \in X$ where $t[A] \neq \perp$:

- $\kappa_e(r, t, A) = \kappa_t(r^K, t[K])$ if $A \in K$,
- $\kappa_e(r, t, A) = \kappa_t(r^Y, t[KY])$ if $A \in Y$ and $t[Y] \neq \perp$, and
- $\kappa_e(r, t, A) = \kappa_t(r^A, t[KA])$ if $t[A] \neq \perp$.

The multilevel database (b^a, κ_t) with tuple-level classification is the *atomic decomposition* of the multilevel database (b, κ_e) with element-level classification. From this definition, we derive the *key classification property* of element-level classification:

6.1 $\kappa_e(r, t, A) = \kappa_e(r, t, A')$ for all $A, A' \in K$, and

6.2 $\kappa_e(r, t, A) = \kappa_e(r, t, A')$ for every $R[Y] \hookrightarrow R'$ in \mathcal{C} , all $A, A' \in Y$, and $t[Y] \neq \perp$.

When $\kappa_e(r, t, A) = \kappa_e(r, t, A')$ for all $A, A' \in Y$, we denote $\kappa_e(r, t, A)$ by $\kappa_e(r, t, Y)$.

From the polyinstantiation property of tuple-level classification, we know that $t[K] = t'[K]$ and $\kappa_t(r^Y, t) = \kappa_t(r^Y, t')$ implies $t = t'$ for all $t, t' \in r^Y$. Similarly $t[K] = t'[K]$ and $\kappa_t(r^A, t) = \kappa_t(r^A, t')$ implies $t = t'$ for all $t, t' \in r^A$. From these properties we derive the *polyinstantiation property* of element-level classification:

7 for every r over $R[X, K]$ in \mathcal{R} , tuples $t, t' \in r$, and attribute $A \in X - K$, we have that $t[K] = t'[K]$, $\kappa_e(r, t, K) = \kappa_e(r, t', K)$, and $\kappa_e(r, t, A) = \kappa_e(r, t', A)$ implies $t[A] = t'[A]$.

From the foreign key security property of tuple-level classification and referential dependencies $R^Y[K] \hookrightarrow R^K$, $R^A[K] \hookrightarrow R^K$ in \mathcal{C}^a , we know that $t[K] = t'$ implies

$\kappa_t(r^K, t') \preceq \kappa_t(r^Y, t)$ for all $t \in r^Y, t' \in r^K$; and $t[K] = t'$ implies $\kappa_t(r^K, t') \preceq \kappa_t(r^A, t)$ for all $t \in r^A, t' \in r^K$. From these properties we derive the *primary key security property* of element-level classification:

8 for every r over $R[X, K]$ in \mathcal{R} and tuple $t \in r$, we have that $\kappa_e(r, t, K) \preceq \kappa_e(r, t, A)$ for all $A \in X - K$ where $t[A] \neq \perp$.

Again from the foreign key security property of tuple-level classification and the referential dependency $R_i^Y[Y] \hookrightarrow R_j^K$ in \mathcal{C}^a , we know that $t[Y] = t'$ implies $\kappa_t(r_j^K, t') \preceq \kappa_t(r_i^Y, t)$ for all $t \in r_i^Y, t' \in r_j^K$. From this property we derive the *foreign key security property* of element-level classification:

9 for every $R_i[Y] \hookrightarrow R_j$ in \mathcal{C} and tuples $t_i \in r_i, t_j \in r_j$ where $t_i[Y] = t_j[K_j]$, we have that $\kappa_e(r, t_j, K_j) \preceq \kappa_e(r, t_i, Y)$.

Our properties 2.1, 6.1, and 8 together form the entity integrity as defined in [3, 6]. Hence our definition of the security semantics of element-level classification provides a semantic justification of entity integrity. Furthermore, our properties 3.1, 3.2, 6.2, and 9 together provide a formal definition and semantic justification of referential integrity in the multilevel relational model.

Figure 6 shows a multilevel database with element-level classification, over the schema of Figure 2. The classification of every element is specified as its superscript. Notice that its atomic decomposition is the multilevel database of Figure 5 with tuple-level classification. Hence the two multilevel databases are semantically equivalent. The null values in Figure 6 have disappeared in Figure 5.

A	B	C	D	E	D'	A'	B'	F
a^l	b^l	c^h	d_1^l	$e_1^{m_1}$	$d_1^{m_1}$	a^{m_1}	b^{m_1}	f_1^h
			d_2^l	$e_2^{m_2}$	$d_2^{m_2}$	\perp	\perp	f_2^h

Figure 6: Element-Level Classification

4.3 Tradeoff

From our definition of the security semantics of element-level classification, we can conclude that tuple-level and element-level classification schemes have exactly the same expressive power, because for every multilevel database with element-level classification, we can find a multilevel database with tuple-level classification that captures exactly the same information in a multilevel state of the world, and vice versa.

But tradeoff does exist between tuple-level and element-level classification schemes when designing a multilevel database with data classification. On one hand, element-level classification is more complicated than tuple-level classification, since classification levels are attached to elements rather than tuples. On the other hand, tuple-level classification requires more complicated schema to capture the same amount of information as element-level classification, making query and update more expensive because the same elementary fact captured by one tuple with element-level classification is captured by several tuples with tuple-level classification.

5 POLYINSTANTIATION INTEGRITY

As an application of the security semantics of element-level classification, we analyze models of polyinstantiation integrity that have been proposed in the literature. As we argued before, null values should not be classified in multilevel databases with element-level classification. With this requirement, our polyinstantiation property 7 is equivalent to the PI-FD property of [11]. Various extensions of this property have been approached, the most representative being the three models discussed in [11], to eliminate intuitively "undesirable" multilevel relations.

The polyinstantiation integrity of the SeaView model[3] has an additional PI-MVD property, which informally says that for every tuple there should be a tuple for every combination of classified non-key attribute values. For the two relations in Figure 7, the first would be allowed but the second would be prohibited. However, it is easy to verify that the two relations are semantically equivalent, because their atomic decompositions are identical. The critical difference between the two relations lies in redundancy rather than semantics. Intuitively, the first relation is more redundant than the second relation, hence the second should be more desirable than the first.

Starship#	Mission	Destination
Enterprise ^l	Exploration ^l	Talos ^l
Enterprise ^l	Exploration ^l	Rigel ^h
Enterprise ^l	Spying ^h	Talos ^l
Enterprise ^l	Spying ^h	Rigel ^h
Enterprise ^l	Exploration ^l	Talos ^l
Enterprise ^l	Spying ^h	Rigel ^h

Figure 7: PI-MVD

The polyinstantiation integrity of the Oakland model[5] has an additional PI-null property, which informally says that any non-key attribute values in two tuples with the same primary key value and classification must either be both null or be both non-null. For the two relations in Figure 8, the first would be allowed but the second would be prohibited. However, it is again easy to verify that the two relations are semantically equivalent, because their atomic decompositions are identical. The critical difference between the two relations lies again in redundancy rather than semantics. Intuitively, the null values in the second relation are redundant because they are implied by the first relation, hence the first is more desirable than the second.

The polyinstantiation integrity of the Franconia model[11] has an additional PI-tuple-

Starship#	Mission	Destination
Enterprise ^l	Spying ^{m₁}	Rigel ^{m₂}
Enterprise ^l	Spying ^{m₁}	⊥
Enterprise ^l	⊥	Rigel ^{m₂}

Figure 8: PI-null

class property³, which informally says that two tuples with the same primary key value, the same primary key classification, and the same least upper bound of attribute value classification, should agree on their non-key attribute values. For the two relations in Figure 9, the first would be allowed but the second would be prohibited. However, it is also easy to verify that the two relations are semantically equivalent, because their atomic decompositions are identical. The critical difference between the two relations lies also in redundancy rather than semantics. Intuitively, the elementary fact “the destination of Enterprise is Rigel” is recorded once in the first relation but twice in the second relation, hence the first is more desirable than the second. Notice that PI-null does not prohibit the second relation.

Starship#	Mission	Destination
Enterprise ^l	Exploration ^l	⊥
Enterprise ^l	Spying ^h	Rigel ^h
Enterprise ^l	Exploration ^l	Rigel ^h
Enterprise ^l	Spying ^h	Rigel ^h

Figure 9: PI-tuple-class

Finally, the two relations in Figure 10 all satisfy the PI-tuple-class property, although the first is redundant while the second is not. It is easy to verify that the two relations are semantically equivalent, since their atomic decompositions are identical. However, the second relation is not necessarily more desirable, because the null value in the second tuple is misleading. By the semantics of null values, it should mean that the mission of Enterprise is unknown to *h*-users, but the elementary fact “Enterprise is on an exploratory mission” should be true for *h*-users according to the first tuple. It has been proposed in

³Contrary to the claim in [11], PI-FD and PI-tuple-class properties are incomparable: neither implies the other.

[11] that the semantics of null values be modified from “no available data at this level” to “no additional data at this level”, in order to accommodate the second relation. This solution not only further complicates the problem of null values, but also violates the principle of standard relational model that all elementary facts about an entity should be collected in a single tuple. For the second relation, the elementary facts on Enterprise for h -users are scattered in two tuples.

Starship#	Mission	Destination
Enterprise ^l	Exploration ^l	Talos ^l
Enterprise ^l	Exploration ^l	Rigel ^h
Enterprise ^l	Exploration ^l	Talos ^l
Enterprise ^l	⊥	Rigel ^h

Figure 10: Redundancy vs Null Value

Hence it is fair to say that, while the essence of the basic polyinstantiation integrity property is to capture functional dependencies in a multilevel state of the world, the essence of various extensions of the basic polyinstantiation integrity property is instead to *eliminate redundancy* in a multilevel database, although no existing proposals fully meet this requirement. In fact, we can show that redundancy cannot be completely eliminated without problematic uses of null values, as evidenced by the second relation of Figure 10. So there cannot be a *perfect* polyinstantiation integrity that both preserves functional dependency and eliminates redundancy. Notice that redundancy is *not* an artifact of the state of the world that a database is to capture, but rather an artifact of the database itself. In comparison, functional dependencies represent general laws of a state of the world. Following the spirit of standard relation model where integrity constraints capture real world semantics, we conclude that polyinstantiation integrity should be designed to represent functional dependencies only, and should not be overloaded with the task of redundancy elimination. In other words, the basic polyinstantiation integrity property is appropriate in capturing polyinstantiation integrity in the multilevel relational model.

6 SUMMARY

We characterized the information in a multilevel state of the world that is captured by a multilevel relational database. Based on the characterization, we formalized the security semantics of tuple-level and element-level data classification schemes. Entity and referential integrity constraints for the multilevel relational model are derived from the security semantics. We also showed that the two data classification schemes are equally expressive, and identified design tradeoffs between the two in developing a multilevel relational database.

Polyinstantiation integrity is critical in the multilevel relational model. Besides capturing the notion of functional dependency, it not only provides the basis of an operational semantics of multilevel relational databases, but also serves as correctness criteria for implementation techniques such as the decomposition and recovery algorithm. Applying our security semantics, we derived the polyinstantiation integrity for element-level classification, and showed that existing approaches to polyinstantiation integrity are inappropriate.

We believe that the controversy surrounding the polyinstantiation integrity, the operational semantics, and even the decomposition and recovery algorithm in multilevel relational databases is primarily due to the lack of a proper security semantics for element-level classification. By formalizing the security semantics, our results supply the foundation for the multilevel relational model, its integrity properties, its operational semantics, and its implementation.

References

- [1] T. F. Lunt, "The True Meaning of Polyinstantiation: Proposal for an Operational Semantics for a Multilevel Relational Database System"; *Proceedings of the Third RADC Database Security Workshop*, June 1990, 26-36.
- [2] T. F. Lunt, "Polyinstantiation: an Inevitable Part of a Multilevel World"; *Proceedings of the Fourth Workshop on the Foundations of Computer Security*, June 1991.
- [3] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley, "The SeaView Security Model"; *IEEE Transactions on Software Engineering* 16:6, June 1990, 593-607.
- [4] T. F. Lunt and D. Hsieh, "Update Semantics for a Multilevel Relational Database System"; *Proceedings of the Fourth IFIP WG11.3 Workshop on Database Security*, September 1990.
- [5] S. Jajodia and R. Sandhu, "Polyinstantiation Integrity in Multilevel Relations"; *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, May 1990, 104-115.
- [6] S. Jajodia and R. Sandhu, "A Novel Decomposition of Multilevel Relations into Single-Level Relations"; *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, May 1991, 300-313.
- [7] S. Jajodia and R. Sandhu, "Toward a Multilevel Secure Relational Data Model"; *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1991, 50-59.
- [8] S. Jajodia, R. Sandhu, and E. Sibley, "Update Semantics of Multilevel Relations"; *Proceedings of the Sixth Annual Computer Security Applications Conference*, December 1990, 103-112.

- [9] J-M. Nicolas and H. Gallaire, "Data Base: Theory vs. Interpretation"; *Logic and Databases*, H. Gallaire and J. Minker (editors), Plenum Press, 1978, 33-54.
- [10] R. Sandhu and S. Jajodia, "Restricted Polyinstantiation or How to Close Signaling Channels without Duplicity"; *Proceedings of the Third RADC Database Security Workshop*, June 1990, 13-25.
- [11] R. Sandhu, S. Jajodia, and T. F. Lunt, "A New Polyinstantiation Integrity Constraint for Multilevel Relations"; *Proceedings of the IEEE Workshop on Computer Security Foundations*, June 1990, 159-165.
- [12] B. M. Thuraisingham, "A Nonmonotonic Typed Multilevel Logic for Multilevel Data/Knowledge Base Management Systems"; *Technical Report MTR-10935*, MITRE, June 1990.

INFERENCE SECURE MULTILEVEL DATABASES

T. Y. Lin

Mathematics and Computer Science
San Jose State University
San Jose, California 95192

1. INTRODUCTION

What is database security? It is not computer security applied to database systems. Multilevel databases with access control are not necessarily secure, because inference threats are rather real in database systems [Hinke88], [Lunt89]. Naturally, we wonder if we should or could stop all possible inference attacks. The answers are that unfortunately we can not and fortunately we need not. The essence of database security research is then to find a practical and acceptable level of inference free database systems. For this, we define the **navigational inference as the process of accessing high data via navigating through legally accessible low data**. Such inference includes the classical inference studied by Hinke and Lunt. We believe that the navigational inference is the "proper and correct" level of inference to be stopped. So we propose the following:

A multilevel data model is **inference secure** if

- (1) it is a Bell LaPadula Model (secure under "MAC"), and
- (2) it is navigational inference free.

In [Lin91], we gave a preliminary report on algebraic inference free (navigational inference free) multilevel relational data models; We showed that the lattice model is an algebraic inference free relational data model. In this paper, we give a full and formal report on the work and some of its generalizations. In fact, we have shown the converse that an algebraic inference free model is the lattice model. This paper is a cumulation of our earlier works on security algebra (lattice model), aggregation, and rigorous applications of Bell LaPadula model to data models.

This research is supported by MDA904-91-C-7048

1.1. Main Results.

A data model consists of data structures, operations and constraints [TsLo82]. A multilevel relational model R consists of multilevel relational structure, relational algebra, and constraints (primary key, foreign key, and ..). Let D be the set of primitive data in R . The set of information (view instances) stored in R is a subset of $P(P(D))$. Let SC be the lattice of security classes. A security map of R assigns each view instance a security class. If the security class of each view instance is the l.u.b. of the security classes of its elements, then the security map is called a **lattice model**. It is easy to see that a security map is a lattice model iff it is a homomorphism which maps the Union operator of R to the l.u.b. operator of SC . If the security class of each tuple or its subtuple is the l.u.b. of the security classes of its elements, then the security map is called a **horizontal lattice model**. A security map is said to be a horizontal homomorphism if the security map is a homomorphism on each tuple of R . One can easily see that a security map is a horizontal lattice model iff it is a horizontal homomorphism. If the aggregation problems of R are all removed, then the security map is called an aggregation free security map.

Theorem 7.2. The aggregation free security map

$$[]: P(P(D)) \text{ -----} \rightarrow SC$$

is a lattice model, or equivalently, a homomorphism which maps the Union operator of R to the l.u.b. operator of SC .

Remark: In [Lin90a, pp. 332] we observed that the minimum set of relational operators (to generate all view instances) consists of Cartesian Product and Union. By using the "pure" set theory, in this paper the minimum set is reduced to a singleton, namely the Union operator.

Hinke studied the inference problems via second paths analysis. A multilevel relational database is said to be second paths inference free if all the second paths have been eliminated. We give such a database an algebraic characterization.

Theorem 7.6. A multilevel relational database is second paths inference free iff the security map is a horizontal lattice model.

In the relational model (not logic based model), one uses the relational algebra to navigate, so the algebraic inference is the navigational inference.

Theorem 9.3. A multilevel relational database is algebraic inference free iff the security map $[]$ is a lattice model.

Corollary 9.4. A multilevel relational database is second paths inference free if its security map is a lattice model.

2. "CLASSICAL INFERENCE"

Example 2.1.

This is our interpretation of Hinke's example [Hink88]. We will be responsible for our interpretation or mis-interpretation.

VISITOR-LOG

Visitor-name	Visitor-company	Contact	Tuple-Class
Peterson	Hughes	John	U

MEETINGS

Room	Time	Project-number	Contact	Tuple-Class
MH123	13:00	SP92745	John	U

CONTRACTS

Project-number	Classification	Tuple-Class
SP92745	Secret	U

CONTRACTORS

Project-number	Company	Tuple-Class
SP92745	Hughes	S

According to [Hinke88], if the join actually exists, then the database has an inference problem.

Path 1 below is a tuple in the join of three relations VISITOR-LOG, MEETING and CONTRACTS.

Path 1:

Peterson--Hughes-- John--MH123-- 13:00--SP92745--Secret

Path 1 is accessible to unclassified users, since it is a join of unclassified tuples. On the other hand, it has a subpath

Path 2:

Hughes ----- SP92745

which is classified as secret in the relation CONTRACTORS. So unclassified users can access high data(Path 2) via the join of low data (Path 1).

The next example is taken from [Lunt89]

Example 2.2. Suppose there is a relation $R(A,B,C)$, where A and B are classified as SECRET and C as TOP_SECRET. Further suppose there is an integrity constraint

$$C = A * B$$

Then a SECRET user can "infer" the TOP_SECRET information C via join operation.

In [Hinke89], Hinke asserted that "while many second paths may exist, it suffices to find just one to perform an inference. If this path is closed, then technique can search for any other second paths. Ultimately, all such second paths must be eliminated to eliminate the inference possibilities from the database."

Our goal is similar to Hinke, however, approaches are totally different. Hinke's approach is geometric in nature, namely, he searches for and closes all possible second paths. Our approach is algebraic in nature, we structure the security classification carefully so that all such second paths are closed from the very beginning (never opened).

3. INFERENCE PROBLEMS AND DATABASE STRUCTURES

Databases are very rich in structures, for examples, relational algebra, data manipulation language (navigational operators), functional dependencies, multi-valued dependencies, semantic networks, first order logic. One can use these structures to compute or derive "new" set of data from "old" set of data. In secure computer systems both "old" and "new" data have security classes. If the security classes of "old" and "new" set of data are not consistent, inference problems arise. In general,

Inference problems exist if the security classifications of data are inconsistent with some database structures.

Logical inference problems: The security classification of a theorem (a derivable formula) in a formal theory should be consistent with its proof. If not, then logical inference problems arise.

Algebraic inference problems: The security classification of "algebraically derivable data" should be consistent with its relational algebraic structure. If not, algebraic inference problems arise.

Navigational inference problems: The security classification of data should be consistent with its navigational operators (which generate navigational paths). If not, navigational inference problems arise.

Most fortunately we found some general solutions for algebraic and navigational inference problems.

4. ALGEBRAIC INFERENCE AND CLASSICAL EXAMPLES

In the literature the terms sensitive data and high data are used interchangeably. We will use, instead of **sensitive** data, the term **high** data. The reason is that the connotation of sensitive data is a reflection of human opinion, while high data is an actual representation or expression of his opinion in computer systems. **Computer systems can not detect sensitive data (they can not read human mind) unless they are classified high.** Sensitive data that are miss-classified consistently as low data are certainly available to low users; we do not call such "mistakes" or "uncovering such mistakes" as inferences.

Let E, F, G and H be relations and

$$E = F * G * H.$$

If the security class of E is strictly higher than that of F, G and H, then there is an inference problem; we will call it **join inference**. More generally, we have the following proposition.

Proposition 4.1. Suppose there is a relational algebraic equation

$$E = f(F, G, H, \dots).$$

An inference exists if

$$[E] > \text{l.u.b. } \{[F], [G], [H], \dots\}$$

where E, F, G, H and ... are relations. Such inference is called **algebraic inference**.

Proof: A low user who are allowed to access low data F, G, H, and ... can infer or derive the high data E by the algebraic expression f. So there is an (algebraic) inference problem.

Proposition 4.2. A database is algebraic inference free if

$$[E] \leq \text{l.u.b. } \{[F], [G], [H], \dots\}$$

is true for **all** relational expression $E = f(F, G, H, \dots)$.

Proof: A low user who is entitled to access data F, G, H, and.. is entitled to access E. So there is no inference problem.

Example 4.3. The Example 2.1 Revisited. Path 1 is a tuple in the following relation (instance)

NEW_RELATION = VISITOR-LOG * MEETING * CONTRACTS.

Note that CONTRACTORS (Path 2 is its tuple) is a "sub-relation" of NEW_RELATION (Path 1 is its tuple). So there is a projection

$$\text{CONTRACTORS} = \text{Proj} (\text{NEW_RELATION})$$

Thus, we have an algebraic relation

$$\text{CONTRACTORS} = \text{Proj} (\text{VISITOR-LOG} * \text{MEETING} * \text{CONTRACTS})$$

Their security classifications satisfy the following inequality

$$[\text{CONTRACTORS}] > \text{l.u.b} \{[\text{VISITOR-LOG}], [\text{MEETING}], [\text{CONTRACTS}]\}$$

Thus users can infer the high data, CONTRACTORS, by the algebraic relation.

5. BELL LAPADULA DATA MODEL (BL-DM)

-- The Data Model As A Bell LaPadula Model.

In Bell Lapadula Model (BLM), an object or a subject is assigned a security class. Now if we apply BLM to database systems, then BLM requires that every object processed by database systems should have security classification. What are the objects processed by databases?

(1) Intentional Objects: They are objects in Data Dictionary, such as, names of attributes, relation schemas, view schemas, relational algebraic expressions, query statements (relational calculus), and constraints. The security class of an intentional object is related to its extension. For example, the security class of an attribute name should be dominated by the security classes of its elements of attribute domain. The security class of a view schema should be dominated by its tuple instances. The security class of an intentional object protects the existence of a high datum.

(2) Extensional Objects: They can be a single element, a tuple, a view or relation instances. In next section, we will reformulate the relational model so that an element is a primitive data, a tuple is a set of primitive data, and a relation or view (instance) is a set of sets of primitive data.

A relation (instance) can be generated from elements by Cartesian product and Union [Lin91a]. Cartesian product produces the tuples, and Union produces the relation instances. In this paper, we represent a tuple as a union of elements. Thus a relation instance is a set of sets of primitive data. We reduce "the generating operations" of relational structures into a single operation, Union. The security classification of extensional objects can then be derived from Union. This give us a consistent way of classifying all the extensional objects. However, in some occasion, we may emphasize the Union within a tuple, then we will term it as **inhomogeneous Union**. In the next section, we will give an exposition on this "new" representation of the relational model.

6. SET REPRESENTATION AND SECURITY CLASSIFICATIONS

In this section, we will show that all relation instances can be generated by the operator, Union, on the primitive data.

6.1. Tuples as Sets

We will reexamine the construct of Cartesian product and interpret the ordered tuples as a set of "inhomogeneous" elements.

Let A_j be attribute domains. Strictly speaking, A_j plays two roles, one is the name of the attribute domains, another is the set itself. We may use $\text{Name}(A_j)$ or $\text{Domain}(A_j)$ to denote one of these two roles whenever there is some danger of confusing.

Let F be a family of attributes. That is,

$$F = \{A_1, A_2, \dots, A_j, \dots\} = \{\text{Name}(A_j) : j = 1, 2, \dots\}$$

Let S be the union of all A_j 's.

$$\begin{aligned} S &= U\{A_j : j = 1, 2, \dots\} = U\{\text{Domain}(A_j) : j = 1, 2, \dots\} \\ &= A_1 \cup A_2 \cup \dots \cup A_j, \dots \end{aligned}$$

The Cartesian Product of A_j 's

$$P = A_1 \times A_2 \times \dots \times A_i \times \dots$$

consists of all possible functions

$$t: F \rightarrow S,$$

where $t(A_j)$ is an element of A_j .

There is a one-to-one correspondence between the function t and its graph:

$$t \longleftrightarrow \{(A_j, t(A_j)) : j = 1, 2, \dots\}$$

The function t is called ordered tuple. The graph is a set of attribute value pair $(A_j, t(A_j))$ [Hsaio70], [Date90].

6.2. Relations and Power sets

Let D , called primitive data, be the set of all attribute value pairs of a database. Then a tuple, as shown above, is a set of attribute value pairs. So tuples are elements of $P(D)$, the power set of D . Since a relation (instance) is a set of tuples. So, a relation instance, as well as a view instance, is an element of $P(P(D))$. A database returns view instances to a user's query. In other words, a database provides information through a set of view instances. Thus information supported by a database is a subset of $P(P(D))$ [Lin91a]. A datum can be interpreted as a view with a

single datum. A tuple is a view with single row. So both D and P(D) can be identified as subsets of P(PD)). The collection of all view instances are the extensional objects.

Example 6.2. Let E be a relation.

E = US-TROOPS

city	CL	troops#	CL
Rome	S	755	S
Athens	S	345	S
London	C	231	C
Berlin	S	500	S
Paris	S	500	S

The relation E is a set of ordered tuples, namely

E = { Rome X 755,
Athens X 345,
London X 231,
Berlin X 500,
Paris X 500 },

where ordered tuples are denoted by Cartesian product X.

We represent these tuples as sets of attribute pairs:

t1={{city, Rome}, {troop#, 755}},
t2={{city, Athens}, {troop#, 345}},
t3={{city, London}, {troop#, 231}},
t4={{city, Berlin}, {troop#, 500}},
t5={{city, Paris}, {troop#, 500}}.

Then E is a set:

E = {t1, t2, t3, t4, t5}

In other words, the relation E is a set of sets. A closely related set, denoted by Set(E), is the union of all tuples:

Set(E) = { (city, Rome), (troop#, 755),
(city, Athens), (troop#, 345),
(city, London), (troop#, 231),
(city, Berlin), (troop#, 500),
(city, Paris), (troop#, 500) }

The Set(E) and E give us basically the same primitive information; they have the same primitive data. Strictly speaking E has more information than Set(E), it has the intermediate structure, namely, tuples, which may carry some information that Set(E) can not carry. For security purpose, they carry the same secrecy labels.

7. SECURITY MAPS FOR THE RELATIONAL MODEL

7.1. Implied Labels

SeaView classifies individual elements, TRW and many others classifies tuples. We would like to point out that either approach give us some form of element level labeling. Suppose we are given an unclassified tuple in which there is no label on any individual element:

MH123 13:00 SP92745 John U

Note that since the tuple is unclassified, an unclassified user can access any element of the tuple. This element level accessibility implies that each element is effectively "unclassified". We will call this **effective class** the **implied label** or **implied security class**. Hence whatever the security classification scheme is, all elements have labels or **implied labels**. So it is legal to assume that in either approach an element level labeling of primitive data is given. That is, a security map

[]: D -----> SC

is given.

7.2. Aggregation Problems

The secrecy semantics of a collection of data often requires us to assign the collection a higher security class. Such problems are referred to as aggregation problems. There are several proposals to solve such problems.

- (1) Hinke: Upgrade some elements in the given aggregate.
- (2) Lunt: Upgrade all elements.
- (3) LDV : Upgrade some elements depending on the past history.
- (4) Lin: There are two parts in this solution. Part one is the solution for a single aggregate: The whole aggregate is classified high, and the internal data can only be seen via aggregate. So effectively, it is equivalent to Lunt's. For Part two, there is an algorithm APR to remove all "redundant" aggregates. The selected aggregates are called marked aggregates.

If we apply Lunt's or Hinke's solution to these marked aggregates, we solve the aggregation problem with minimum upgrading. In the rest of this paper, we assume that all aggregation problems are solved by this Lin-Lunt or Lin-Hinke methods. So all databases have no aggregation problem.

7.3. Algebraic Classifications Maps

A security map on primitive data

[]: D -----> SC

is given, say by DoD.

Let CO be the set of all information stored in the database; it is the set of all view instances [Lin91a]. Now we will extend the security map to

$$H[]: CO \longrightarrow SC,$$

which is defined by

$$H[X] = 1.u.b.\{[x]: x \text{ in Set}(X)\}$$

Intuitively, it is the 1.u.b. of the labels of all primitive data appeared in X. The H in H[] denotes both the High water mark and the algebraic Homomorphism determined by [].

Proposition 7.1. H[] is a homomorphism from a subset CO of P(P(D)) to SC which maps the Union operator to the 1.u.b. operator of SC.

Proof: Let X and Y be two elements in CO. Then

$$\begin{aligned} H[X \cup Y] &= 1.u.b.\{[z]: z \text{ in Set}(X \cup Y)\} \\ &= 1.u.b.\{[z]: z \text{ in Set}(X) \cup \text{Set}(Y)\} \\ &= 1.u.b.\{[z]: z \text{ in Set}(X)\} \cup 1.u.b.\{[z]: z \text{ in Set}(Y)\} \\ &= H[X] \# H[Y], \end{aligned}$$

where # is the 1.u.b. operator in SC.

Definition 7.2. The security map H[] is called the lattice model or the algebraic classification map.

Next we will investigate general security maps which are defined on CO. Let an arbitrary security map be

$$Sec: P(P(D)) \dashrightarrow SC$$

From the secrecy semantic, the security class of a subset should be dominated by that of the whole set. From BLM, all information should be classified. More precisely, the security map Sec should satisfy the following constraints:

(1) Monotonicity constraint: Let A, B be two elements in P(P(D)).

$$Sec(A) \geq Sec(B) \text{ if Set}(B) \text{ is a subset of Set}(A)$$

(2) Totality constraint: Sec is defined on CO(all view instances).

The constraint (1) is implied by the secrecy semantics, (2) is implied by Bell LaPadula Model.

In Section 7.2, we have assumed that all aggregation problems have been removed (otherwise, there are inferences via aggregates). So we have the following constraint.

(3) Aggregation free constraint: Let X be an element of $P(P(D))$.

$$\text{Sec}(X) = \text{l.u.b.} \{[x]: x \text{ in } X\},$$

where $\text{Sec}(x)$ is denoted by $[x]$.

One can easily show that (3) implies (1): If X is a tuple, then

$$\text{Sec}(X) = \text{l.u.b.} \{[x]: x \text{ in } \text{Set}(X)\}.$$

If X is a view instance, then

$$\begin{aligned} \text{Sec}(X) &= \text{l.u.b.} \{[x]: x \text{ are tuples in } X\} \\ &= \text{l.u.b.} \{[x]: [x] = \text{l.u.b.} \{[t]\}\} \\ &= \text{l.u.b.} \{[t]: t \text{ in } \text{Set}(X)\} \end{aligned}$$

So

$$\text{Sec}(B) \leq \text{Sec}(A) \text{ if } \text{Set}(B) \text{ is subset of } \text{Set}(A)$$

Proposition 7.2. The security map, which satisfy the three constraints,

$$\text{Sec}: P(P(D)) \text{ ----(partial)----> } SC$$

is the lattice model $H[]$ (the algebraic classification map).

Proof: Let X be an element in $P(P(D))$ (i.e., a view instance).

$$X = \{t_1, t_2, \dots, t_n\}$$

where $t_i = \{e_{i1}, e_{i2}, \dots\}$ $i = 1, 2, \dots$ are tuples, where e_{ij} are data in D . By (3), ($\text{Sec}(x)$ is denoted by $[x]$)

$$\begin{aligned} [t_i] &= \text{l.u.b.} \{[e_{i1}], [e_{i2}], \dots\} \\ &= [e_{i1}] \# [e_{i2}] \# \dots \end{aligned}$$

Again, by (3)

$$\begin{aligned} \text{Sec}(X) &= [X] = \text{l.u.b.} \{[t_1], [t_2], \dots, [t_n]\} \\ &= [t_1] \# [t_2] \# \dots \# [t_n] \\ &= ([e_{11}] \# [e_{12}] \# \dots) \# ([e_{21}] \# [e_{22}] \# \dots) \\ &\quad \# ([e_{31}] \# [e_{32}] \# \dots) \dots \\ &= \text{l.u.b.} \{[e_{ij}]: e_{ij} \text{ in } \text{Set}(X)\} \\ &= H[X] \end{aligned}$$

Since this is true for any X in CO , so we have proved that the two security maps are equal.

$$\text{Sec} = H[]$$

In other words, all security maps which satisfy the three constraints are precisely the algebraic classification map (the lattice model).

7.4. Second Paths Inference Free Security Maps

Hinke uses second paths to analyze inferences. We will show that "second paths inference free" databases are "horizontal" lattice model.

Let a security map

$$S: CO \text{ -----} \rightarrow SC$$

be given, where CO is a subset of $P(P(D))$. Suppose that we have found all the second paths for all possible pairs of data in D [Hink89]. And suppose that each second path is closed up by upgrading some elements in D . Then we have a new security map

$$S': CO \text{ -----} \rightarrow SC.$$

in which there is no second path inference. Let the restriction of S' to D be $[]'$. Then we have several propositions.

Proposition 7.3. Let A, B be two elements (tuples) in $P(D)$. Then

$$S'(A) \geq S'(B) \text{ if } B \text{ is a subset of } A$$

Proof: Let B be a subset of A . Let UN be the maximal element in $P(D)$ that contains A . Then the monotonicity constraint holds for all subsets in UN . Otherwise, one can easily found a second path inference; This can be accomplished by mimic the arguments of Hinke's Example 2.1. This is true for all such UN 's in $P(D)$.

The conclusion of 7.3 is called the horizontal (tuple) monotonicity constraint.

Proposition 7.4. Let X be an element of $P(D)$.

$$S'(X) = \text{l.u.b. } \{[x]: x \text{ in } X\},$$

Proof: Assume $X = \{x_1, x_2, \dots\}$ is an aggregate. Then we can infer X by visiting x_1, x_2, \dots in this order. So aggregation free constraint (3) is satisfied for element in $P(D)$.

The conclusion of 7.4 is the horizontal (tuple) aggregation free constraint.

Definition 7.5. A security map which satisfies the horizontal aggregation free constraint for each tuple or its subtuples is called a horizontal lattice model.

Theorem 7.6. A multilevel relational database is second paths inference free iff its security map is a horizontal lattice model.

Proof: Assume $\{x_1, x_2, \dots, y\}$ is a second path for the direct path $\{x_1, y\}$. Then the maximal tuple UN that contains $\{x_1, y\}$ is not horizontal aggregation free. Hence the security map is not a

horizontal lattice model. Conversely, if $\{x_1, y\}$ has label higher than that of $\{x_1, x_2, \dots, y\}$, then there is a second path inference.

Remark: Let $X = \{t_1, t_2, \dots\}$ be an element in $P(P(D))$, where t_i are tuples. If $[X] > \text{l.u.b.}\{[t_1], [t_2], \dots\}$, then X is a vertical aggregation.

Definition 7.7. A security map which satisfies the vertical aggregation free constraints for all view instances is called a vertical lattice model.

Theorem 7.8. If the security map Sec is a horizontal and vertical lattice model, then Sec is the lattice model.

Proof: Let $X = \{t_1, t_2, \dots\}$ be a set of tuples. By assumption, each t_i is a horizontal lattice model. So

$$\text{Sec}(t_i) = \text{l.u.b.}\{[e_{i1}], [e_{i2}], \dots\}$$

where $t_i = \{e_{i1}, e_{i2}, \dots\}$, and each e_{ij} is an element in D .

$$\text{Sec}(X) = \text{l.u.b.}\{[t_1], [t_2], \dots\}$$

$$\begin{aligned} &= ([e_{11}] \# [e_{12}] \# \dots) \# ([e_{21}] \# [e_{22}] \# \dots) \\ &\quad \# ([e_{31}] \# [e_{32}] \# \dots) \# \dots \\ &= \text{l.u.b.}\{[e_{ij}] : e_{ij} \text{ in } \text{Set}(X)\} \\ &= H[X] \end{aligned}$$

In other words, the Sec is the lattice model.

7.5. Examples

Example 7.9. (Continue from Example 6.2.)

The security class of an element is that of the corresponding attribute value pair, for example,

$$[\text{Rome}] = [(\text{city}, \text{Rome})] = S.$$

The security class of a tuple is the l.u.b. of the security classes of its elements, for examples,

$$[t_1] = [(\text{city}, \text{Rome})] \# [(\text{troop\#}, 755)]$$

$$[t_2] = [(\text{city}, \text{Athens})] \# [(\text{troop\#}, 345)],$$

where $\#$ is the l.u.b. operator of the security lattice SC .

The security class of a relation is the l.u.b. of the security classes of its tuples

$$\begin{aligned} [E] &= [t_1] \# [t_2] \# [t_3] \# [t_4] \# [t_5] \\ &= [(\text{city}, \text{Rome})] \# [(\text{troop\#}, 755)] \# \end{aligned}$$

```

[(city, Athens)] # [(troop#, 345)] #
[(city, London)] # [(troop#, 231)] #
[(city, Berlin)] # [(troop#, 500)] #
[(city, Paris)] # [(troop#, 500)].

```

= [Set(E)]

If there is no regularity in the assignment of security classes to every possible subsets of E, it is an exponential problem [Lin89a]. Since [] is a homomorphism, this problem has been avoid. Note that **careless assignment leads to violations of monotonicity constraint and create inference problems.**

Example 7.10. In Hinke's example, there are two inference paths. We can represent paths by collections of data, namely,

Path 1 = {Peterson, Hughes, John, MH123, 13:00, SP92745, Secret}

We should use attribute value pairs, for example, Peterson should be (person, Peterson). However, in this example, the attributes are obvious, so we suppress them from our presentation. This path is joined by unclassified data, so

[Path 1] = U

However, there is a second path from CONTRACTOR table

Path 2= {Hughes, SP92745}

and its security class is

[Path 2]= S

This security map violates the monotonicity constraint of our model. Therefore Hinke's example can not occur in our model. To close up the inference path, Hinke has to upgrade some of the tuples which contains either Hughes or SP92745.

Analysis: Although TRW only classifies tuples, there is an implied security class for each element. Let us examine the VISITOR-LOG and MEETING tables. The implied security classes of

(2) 'Hughes' and 'SP92745' are U.

On the other hand, the label of Path 2 = {Hughes, SP92745} is S. So Path 2 is an aggregation. Using either Hinke's or Lunt's method we can reclassify the data, so the database become a legal member of our model. To avoid over classifications, we upgrade the element labels, and the tuple label will be the l.u.b. First, we assume that each element receives an implied label.

Case 1. Lunt's upgrading: All elements in Path 2 will be upgraded.

VISITOR-LOG

Visitor-name	Visitor-company	Contact	Tuple-Class
Peterson U	Hughes S	John U	S (was U)

MEETINGS

Room	Time	Project-number	Contact	Tuple-Class
MH123 U	13:00 U	SP92745 S	John U	S (was U)

CONTRACTS

Project-number	Classification	Tuple-Class
SP92745 S	Secret U	S (was U)

CONTRACTORS

Project-number	Company	Tuple-Class
SP92745 S	Hughes S	S

Case 2. Hinke's upgrading: We have two choices. For this particular example, SP9274 will be the right one to be upgraded. VISITOR-LOG is the same as original.

VISITOR-LOG

Visitor-name	Visitor-company	Contact	Tuple-Class
Peterson U	Hughes U	John U	S (was U)

Other tables are the same as case 1.

Now with this new security classes, the database is algebraic inference free.

However, users might find that his data are over-classified. Note that over-classification is a result of the aggregation problem, not the inference. In this case, users may have to decide between over-classification or inference insecure.

8. RELATIONAL OPERATORS ON BL-DM

Let $R(A_1, A_2, \dots)$ be a relation instance. Let $R(A_{k_1}, A_{k_2}, \dots)$, be a subrelation, where A_{k_1}, A_{k_2}, \dots is a subsequence of A_1, A_2, \dots

(1) Projection

P: $R_1 = R(A_1, A_2, \dots) \rightarrow R_2 = R(A_{k1}, A_{k2}, \dots)$

be a projection. The security classes of these relations satisfy

$$[R(A_1, A_2, \dots)] \geq [R(A_{k1}, A_{k2}, \dots)]$$

On The Surface P is a downward flowing operation.

(2) Selection

S: $R_1(A_1, A_2, \dots) \rightarrow R_2(A_1, A_2, \dots)$

be a selection, where R_2 is a subrelation (instance) of R_1 . The security classes of these relations satisfy

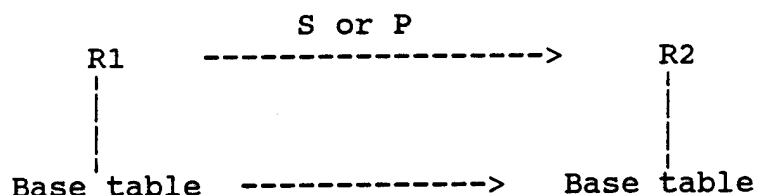
$$[R_1(A_1, A_2, \dots)] \geq [R_2(A_1, A_2, \dots)]$$

Again the selection S is a downward flowing operation.

Therefore both S and P are "illegal" operations. So S and P have to be "trusted subjects". SeaView solved this by its architecture.

8.1. SeaView type solution

In SeaView, the data is stored in base relations, so both R and S are views. The operation S does not transfer data from R to S, it transfers data from base relations of R to base relations of S. So S transfers data "horizontally" (same level of security).



Therefore SeaView's architecture is very important for multilevel data model. Without SeaView's architecture, all relational operations are trusted subjects. Hence their codes has to be trusted too. (Note that trusted codes is not trusted subjects). Such a large trusted codes may become impossible. Some commercial multilevel database systems which are developed without SeaView type architecture may have difficult to show that their models satisfy the security requirements set forth by Bell LaPadula Model.

9. ALGEBRAIC INFERENCE FREE DATA MODELS

Proposition 9.0. $[Set(R)] = [R]$

Proof: Let $R = \{t_1, t_2, \dots\}$ be a set of tuples. Let $t_1 = \{a_1, a_2, \dots\}$

Since $[]$ is a homomorphism, we have

$$[t1] = [a1] \# [a2] \# \dots$$

so the security class of R ,

$$\begin{aligned} [R] &= [t1] \# [t2] \# \dots \\ &= ([a1] \# [a2] \# \dots) \# (\dots) \\ &= [\text{Set}(R)] \end{aligned}$$

(A1) Cartesian product $R1 \times R2$.

Observe the following relation:

$$\text{Set}(R1 \times R2) = \text{Set}(R1) \cup \text{Set}(R2)$$

Proposition 9.1. $[R1 \times R2] = [R1] \# [R2]$

Proof: $[R1 \times R2] = [\text{Set}(R1 \times R2)] = [\text{Set}(R1) \cup \text{Set}(R2)]$
 $= [\text{Set}(R1)] \# [\text{Set}(R2)] = [R1] \# [R2]$.

(A2) Union

Proposition 9.2. $[R1 \cup R2] = [R1] \# [R2]$

The same proof for the Cartesian product works here too.

(A3) Intersection

$$[R1 \cap R2] \leq \text{l.u.b } \{[R1], [R2]\}$$

(A4) Difference

$$[R1 \setminus R2] \leq [R1]$$

(A5) Divide

$$[R1 / R2] \leq \text{l.u.b } \{[R1], [R2]\}$$

(A6) Projection

$$[\text{Proj } R1 \text{ to } \dots] \leq [R1]$$

(A7) Join

$$[R1 * R2] = [R1 \times R2] = [R1] \# [R2]$$

(A8) Selection

$$[\text{Select } R1, \text{ where WFF}] \leq [R1]$$

These analyses lead us to conclude the following.

Theorem 9.3. A multilevel relational database is algebraic inference free iff the security map $[]$ is a lattice model.

Corollary 9.4. A multilevel relational database is second paths inference free if its security map is a lattice model.

Proof of 9.3: To simplify our narration, we will say that the resulting new relations of any relation operations is an output (of the operation). The original relations before the operation is called the input(of the operations).

To prove this theorem, we only need to show that the security class of an output is always less than or equal to the security class of an input, that is,

$$[\text{output}] \leq [\text{input}]$$

An input may have several relations, the security class of an input is the l.u.b. of all these input relations.

The results in A1 - A8 confirm the inequality for every relational operation. So we have proved that it is algebraic inference free.

Conversely, an algebraic inference free data model implies that it is aggregation free (since union is a relational operator). An aggregation free data model has the lattice model as its security map.

Let us restate the theorem in other forms.

Corollary 9.5. Let

$$E = f(F, G, \dots)$$

be an expression of relational algebra. Then

$$[E] \leq \text{l.u.b. } \{[F], [G], \dots\}$$

Verbally, the theorem concludes that

$$[\text{Relational operations on } F, G, \dots] \leq \text{l.u.b. } \{[F], [G], \dots\}$$

10. NAVIGATIONAL INFERENCE

Given a multilevel database system DBS (not necessary relational). Let Op_1, Op_2, \dots be the navigational operators in its data manipulating language. Navigational operators do not involve updating, deletion or insertion.

If the security class of the output of a navigational operator, say

Opi, is strictly greater than that of its input, then there is a navigational inference. Or more formally

Proposition 10.1. Suppose for some k

$$[\text{input of Opk}] < [\text{output of Opk}].$$

Then there is a navigational inference.

Proof: A lower user inputs some low data to the Navigational Operator Opk. By assumption Opk produces some high data to the lower user. In other words, the low user can derive or infer the high data by navigation. This is a navigational inference.

Example 10.2. The navigational operators of Information Management System of IBM are GN (get_next), GNP, GNH, GNPH.

Proposition 10.3. A multilevel database is navigational inference free, if $[\text{output of Opi}] \leq [\text{input of Opi}]$ for all i.

Proof: Since the security class of the output of any navigational operator is always lower than that of its input data, a low user can not infer any high data via navigational operators. There is no inference problem.

11. THE TOWER OF POWER SETS

If we identify x with the singleton {x}, D is a subset of P(D). We will embed D to P(D), similarly P(D) to P(P(D)), and so forth. A finite sequence D, P(D), P(P(D)),... (with all the identifications) is a monotonic increasing sequence of sets. The nth terms will be called the tower of power set of order n. Let PD be the union (direct limit) of this sequence. Then D, P(D), P(P(D)),... are subsets of PD.

Definition 11.1. The set PD is called the tower of power sets. The nth term is called the tower of power set of order n, in short, order n power sets, PDn.

Proposition 11.2. An instance of a data model corresponds to an element of PDn for some n.

Example 11.3. An instance of a relational database corresponds to an element of PD2 or a subset of PD1.

Each relation, say E is a set of tuples and a tuple is a set of primitive data.

Example 11.4. An instance of object oriented database corresponds to an element of PDn for some n.

11.1. Set(E)

A subset in PD is a finite tower of primitive data. It is convenient to express the finite tower as subset of D. Roughly, we are taking the "union" of the tower. In Example 6.2, we give an example of Set(E). We will proceed more formally here.

Let E be an element in D

$$\text{Set0}(E) = \{E\}$$

Roughly, Set0(a primitive datum) is the singleton of the datum.

Let E be an element in $P(D)=PD_1$, we define

$$\text{Set1}(E) = E$$

Example 11.1. Let E be a tuple, say $t1 = \{(city, Rome), (troop\#, 755)\}$

$$E = t1 = \{(city, Rome), (troop\#, 755)\},$$

$$\begin{aligned}\text{Set1}(E) &= t1 \\ &= \{(city, Rome), (troop\#, 755)\},\end{aligned}$$

Let E be an element in $P(P(D))=PD_2$, then

$$\text{Set2}(E) = \text{Union} \{ x: x \text{ is an element in } \text{Set1}(E) \}$$

Example 11.2. Let E be the relation in Example 6.2.

$$E = \{t1, t2, \dots, t5\}$$

$$\begin{aligned}\text{Set2}(E) &= \text{Set1}(t1) \cup \text{Set1}(t2) \dots \cup \text{Set1}(t5) \\ &= t1 \cup t2 \cup \dots \cup t5 \\ &= \{ (city, Rome), (troop\#, 755), \\ &\quad (city, Athens), (troop\#, 345), \\ &\quad (city, London), (troop\#, 231), \\ &\quad (city, Berlin), (troop\#, 500), \\ &\quad (city, Paris), (troop\#, 500) \}\end{aligned}$$

In general, let E be an element in $PD(i+1) = P(PD_i)$, then

$$\text{Set}(i+1)(E) = \text{Union} \{ \text{Set}_i(x): x \text{ is an element } E \}$$

We will simply and vaguely use Set for Set_i for any i. Its meaning should be clear from its context.

11.2. Security Maps for The Tower.

Let D be the primitive data (attribute value pairs) of a database. Let SC be the lattice of security classes. Primitive data are the data that can not be derived by other data. So their security classes have to be given by human. Therefore we assume that the

security labels of the primitive data are given, that is, the security map

$$[]_0: D \text{ -----} \rightarrow SC$$

is given, where $[]_0$ is called 0th order security map.

BLM requires every extensional object has to have a security labels, so we need to extend this map to a subset of PD, namely, a partial map

$$(1) \quad []_\$: PD \text{ ----}(\text{partial map})\text{-----} \rightarrow SC.$$

where $[]_\$$ is the ultimate security map (of infinite order).

We will extend the security map $[]_0$ level by level.

Step 1: The label of a subset is the l.u.b. of the labels of its elements:

$$[]_1: P(D) \text{ ----}(\text{partial map})\text{-----} \rightarrow SC$$

Step 2: The label of a set of subsets is the l.u.b. of the labels of its subsets:

$$[]_2: P(P(D)) \text{ ----}(\text{partial map})\text{-----} \rightarrow SC$$

Step i : Similarly, we can define $[]_i$ for any i , $i=1,2,\dots$

Final Step: Note that D , $P(D)$, $P(P(D))$, and are all identified to some subsets of PD. Taking the union (direct limit) of all the previous $[]_i$'s, we have our final security map:

$$[]_\$: PD \text{ ----}(\text{partial map})\text{-----} \rightarrow SC$$

Since it is a finite tower, note that $[]_\$$ is equal to some $[]_i$ for some finite i .

Proposition 11.4. $[]_\$(X) = []_1(\text{Set}(X))$, for all X in PD.

12. NAVIGATIONAL INFERENCE FREE MULTILEVEL DATA MODELS

In this section, we will investigate the navigational inferences on multilevel databases, especially on MLS-OODB (multilevel object oriented databases). Some MLS-OODB's are based on the assumption that the security classification of every object is given. Such an assumption is not legitimate in the sense that its construction is not trivial. To see my point, let us assumed that a multilevel database application is modeled by both MLS-RDB (multilevel relational database) and MLS-OODB. Let D be the primitive data. An instance of an object in MLS-OODB corresponds to an element in PD.

Thus the (extensional) security map of a MLS-OODB is a partially defined function

S: PD -----> SC

where SC is the poset of security classes and PD is the tower of power sets. The secrecy semantics requires that S is monotonic. **The map S that satisfies this monotonic constraint is called a compatible security map.** Some authors simply assume that such a map exists and it satisfies the monotonicity constraint. With all the semantic constraints on MLS-OODB, such compatible security map is not easy to construct. We believe that a legitimate MLS-OODB should have an algorithm to construct such security map. Another group of authors simply assume a security map exists without any compatibility assumptions. For such MLS_OODB, there will be a lots of inference channels. An inference secure MLS-OODB has to have a compatible security map. The fundamental question is whether such a compatible security map exists. We believe that the security map []\$ constructed is one such map for all extensional objects in MLS-OODB.

13. CONCLUSIONS

The results are somewhat surprising. But it is a natural consequence of the theory of security algebra. Note that in security algebra, a derived data is assigned a security class by its algebraic relation. So there is no inconsistency in security classifications among variables/data of relational algebraic equations. Therefore there is no algebraic inference. In literature, aggregation and inference problems are separated, however, from our point of view the two notions are in one. A good solution for aggregation problems always implies a good solution on inference problems. Finally we should stress that we have no claims on logical inferences.

References

- [Date86] C. J. Date, An introduction to Database Systems, Addison-Wesley, Reading, MA., 1986
- [Denn76] D. E. Denning. "A Lattice Model of Secure Information Flow", Communications of the ACM, Vol. 19, No. 5, May 1976, pp. 236 - 243.
- [GaMiNi84] H. Gallarie, J. Minker, and J. Nicolas, Logic and Databases: A Deductive Approach, Readings in Artificial Intelligence and Databases, ed. J. Mylopoulos and M. Broodie, 1988, pp.231-247.
- [Hink88] T. H. Hinke. Inference Aggregation Detection in Database Management Systems, Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988.

[Hink89] T. H. Hinke. Database Inference Engine Design Approach, Database Security: Status and Prospects II, edited by C. E. Landwehr, North Holland, 1989.

[Hsiao70] David Hsiao, A Formal System for Information Retrieval from Files, CACM, Vol 13, 1970, pp. 67-73.

[Lin89a] T. Y. Lin, A Generalized Information Flow Model and Role of System Security Officer, Database Security: Status and Prospects II, edited by C. E. Landwehr, North Holland, 1989.

[Lin89b] T. Y. Lin, L. Kerschberg, and R. Trueblood, Security Algebra and Formal Models, Database Security: Status and Prospects III, edited by C. E. Landwehr and D. Spooner, North Holland, 1990. Also Proceedings of IFIP WG11.3 Workshop on Database Security September 5-7, 1989.

[Lin89c] T. Y. Lin, Commutative Security Algebra and Aggregation, Proceedings of Second RADC Workshop on Database Security, 1989.

[Lin90a] T. Y. Lin, Multilevel Database, Aggregated Security Algebra, Database Security: Status and Prospects IV, edited by S. Jajodia and C. E. Landwehr, North Holland, 1991. Also Proceedings of IFIP WG11.3 Workshop on Database Security September 18-21, 1990.

[Lin90b] T. Y. Lin. Numerical Measure on Aggregations, Proceeding of 6th annual Application Computer security conference, Dec 3-7, 1990.

[Lin91a] T. Y. Lin, "Inference" Free Multilevel Data Models, Proceedings of Fourth RADC Workshop on Database Security, 1991.

[Lunt89] T. F. Lunt. Aggregation and Inference: Facts and Fallacies, Proceedings of 1989 IEEE Symposium on Security and Privacy, 1989.

[Morg87] Mathew Morgenstern. "Security and Interference in Multilevel Database and Knowledge-base Systems," ACM International Conference on Management of Data (SIGMOD-87), May 1987.

[Morg88] Mathew Morgenstern. "Controlling Logical Inference in Multilevel Database Systems, Proceedings of 1988 IEEE Symposium on Security and Privacy, 1988.

[TsLo82] D. C. Tsichritzis and F. H. Lochovsky, Data Models, Prentice-Hall, Englewood Cliffs, N.J., 1982

A Specification Methodology for User-Role Based Security in an Object-Oriented Design Model*

Experience with a Health Care Application

T.C. Ting, S. A. Demurjian, and M.-Y. Hu
Computer Science and Engineering Department
Box U-155, 260 Glenbrook Road
The University of Connecticut
Storrs, Connecticut 06269-3155

Abstract

Mandatory access control emphasizes the security classification of data and establishes privileges for users based on their ability to read and write data at different levels. This is contrary to discretionary access control (DAC), where the privileges are assigned to users based on their particular needs within the application. User-role based security (URBS) has been proposed as a means to support DAC and considers the responsibilities of users within the application as the guiding factor for determining privileges. Our recent work has explored URBS for an object-oriented design model via a definitional and analytical framework, where security designers can define the different user roles, assign privileges, and evaluate their established security policy against the application's intended requirements. This paper extends our previous work in three ways. First, we offer a set of new techniques to more fully describe user roles and their capabilities. Second, we present a set of new analysis techniques that allow security designers to verify the consistency of their user-role definitions, by automatically alerting the designer when conflicts have been detected among privileges. Finally, we demonstrate that the new definition and analyses techniques form a specification methodology for URBS. To serve as a basis for our discussion throughout this paper we utilize a health care application.

1 Introduction

Mandatory access control (MAC), the reading and writing of data by individuals based on their authorized security clearance level, is the main approach taken in multilevel secure database systems [8,9,13,16,19,22,25,26] using the Bell and Lapadula security model [2]. A complementary approach, *discretionary access control (DAC)*, uses a richer

*The work of S. A. Demurjian and M.-Y. Hu is partially supported by grant IRI-8902755 from the National Science Foundation.

set of access modes that are specific to the particular types or categories of information to those individuals with a need-to-know for the information. Unlike MAC, DAC is closely linked to the requirements of the database application. To support DAC, *user-role based security (URBS)* has been proposed [18,24,27,28], and emphasizes the responsibilities of the end-users within the application when assigning privileges. The relationship between DAC, URBS, and MAC is shown in the top portion of Figure 1. In the figure, the request by the user to access the database must first pass through the identification and authentication (I & A) control, before proceeding on through DAC and MAC to the database, requiring that both policies are satisfied before any response is provided.

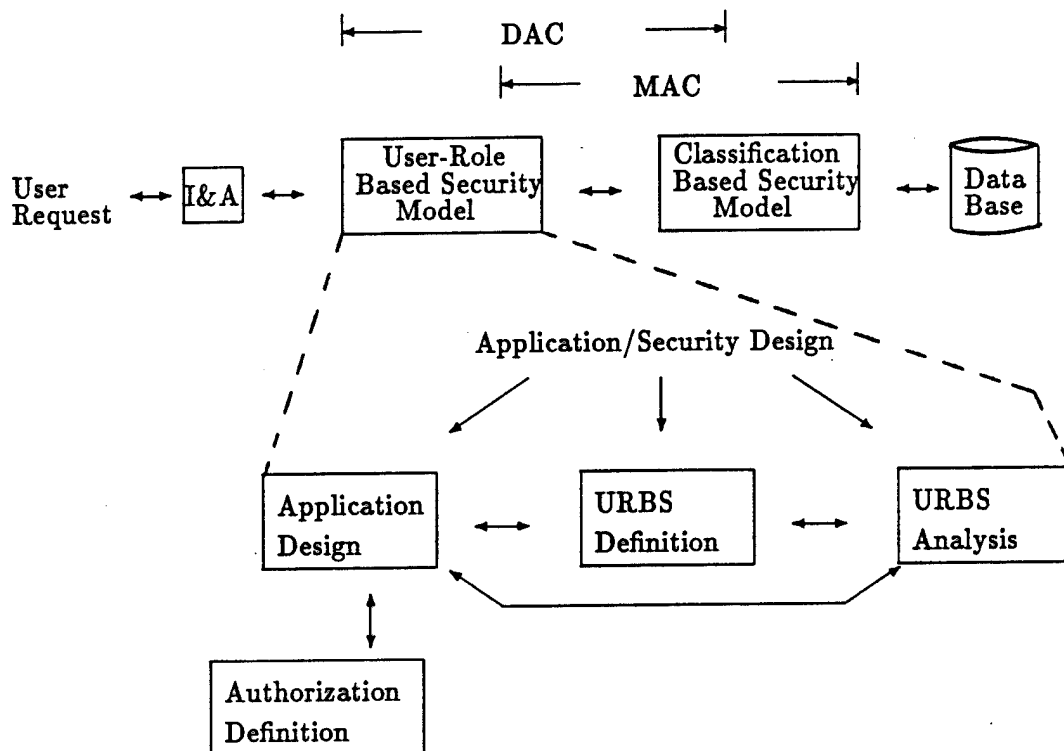


Figure 1: An Overview of Security.

Figure 1 also contains the expansion of the URBS model to include the steps taken by both the application and security designer in developing a specific instance of a URBS model, which has been the focus on our ongoing research effort [6,7,29,30]. We have divided this process into four parts. First, the application's design (i.e., the requirements via the model/schema of the underlying database system) must be understood and considered as an integral aspect of the overall security specification process. Then, URBS can be defined under the guidelines and restrictions of the URBS model. Coincident with this definition process is a set of analysis techniques that allows designers to understand, evaluate, refine, and redefine their security specifications against their application's intended security requirements. The interactions between these first three parts are both iterative and incremental. Once the URBS has been completely defined, authorization can commence, with the security designer granting and revoking, rights

and privileges to individuals based on their user roles. Interactions with the I & A process are also required, to establish the allowable privileges for an individual based on his/her authorization. We have chosen an object-oriented approach for representing the application schema and characterizing its security requirements.

Specifically, we have examined the incorporation of URBS into an object-oriented design model in a manner that both utilizes and is consistent with the object-oriented paradigm and its precepts, principles, and philosophies [6,7,29,30]. A major aspect of the object-oriented paradigm is the separation of the hidden implementation from the public interface, which allows changes to be relatively transparent. It is through the public interface (i.e., the methods which are defined) that users access the object type (i.e., call methods), and hence implicitly access the data that is hidden. The single public interface is shared by all of the users of the object type. Our work has sought to extend the public interface concept, so that different individuals based on their user roles are assigned different subsets of the public interface at different times. This assignment of methods to user roles obscures the data and its access, but is offset by two factors: one, the obscuring is consistent with object-oriented philosophy; and, two, the security designer can focus on abstract concepts (the methods) rather than detailed information (the data). In this process of assigning methods to roles, the security designer is able to review rights (based on his/her understanding of the application's security requirements), to determine the privileges that should be granted.

To support the definition of URBS, our previous research has provided a mechanism that allows designers to identify and organize the different types of users and their roles in an application via a hierarchy. We extend this effort to emphasize the definitional aspects of DAC for characterizing user roles and their responsibilities. We have developed a new set of definitional techniques that allow the security designer to more fully describe user roles and their capabilities within the application. Throughout the definition process, a set of new analysis techniques are available for verifying consistency and identifying conflicts within user-role definitions. Some of these techniques are designer initiated, while others automatically provide feedback to the designer when his/her action will introduce an inconsistency into the URBS definition. To assist the designer in the definition and analyses of the security privileges for the application, a specification methodology for URBS is promoted. Through the methodology, a more precise characterization of the user roles and their security privileges can be attained. To serve as a basis for our discussion throughout this paper we utilize a health care application. Health care information management and its security control are a growing concern, and have received much attention in the national media.

The remainder of this paper is organized into four sections. In Section 2, we present background information on the concepts of our object-oriented design model and an example using health care databases. In Section 3, we examine the definition process of a hierarchy of user roles and the analysis techniques that promote more accurate and complete design. Section 4 discusses the specification methodology including its implications and reflects on its applicability to the advanced needs of applications such as health care. Finally, Section 5 summarizes the paper and indicates ongoing research.

2 Background Concepts

2.1 An Object-Oriented Design Model

In this section, we present the characteristics and features of an object-oriented design model, utilized as a basis for our work, and influenced by work on model and paradigm concepts [1,14,32,34], and our own effort in the development of object-oriented modeling tools [4,5,11]. In object-oriented models, an object has a unique identifier, and encapsulates a state (i.e., the values for the attributes of the object) and a behavior (i.e., the set of methods that operate on the attributes). When objects share a similar state and behavior, an object type is defined:

Definition 1: An *object type*, OT , is defined using a three tuple: $(OTName, \mathcal{D}, \mathcal{M})$ where $OTName$ uniquely identifies the OT , \mathcal{D} is a set of private data for the OT , and $\mathcal{M} = \{\mathcal{HM}, \mathcal{PM}\}$ is a set of hidden and public intra-type methods that can only be applied to the instances of OT .

Definition 2: \mathcal{D} is defined as a set of unique attribute name, attribute type pairs, where the type may be primitive (e.g., integer, float, character, etc.) or complex (another object type).

Many of our assumptions for an object-oriented design model have their origins in abstract data types [17]. This is especially true for our interpretation of information hiding:

Definition 3 : To support information hiding, an OT is partitioned into a *hidden private implementation*, (HPI), and a *potential public interface*, (PPI). \mathcal{D} and \mathcal{HM} are part of HPI , while $\mathcal{PM} \equiv PPI$.

This definition extends the information hiding concept to support database security. The public interface serves as the basis for making the OT available to different users. Hence, the methods of the public interface have the potential to be made public, at the discretion of the security designer. The distinction between public and private methods must be maintained, since it is the cornerstone of object-oriented concepts. Effectively, we are prohibiting the possibility that any private methods can ever be made available for general use.

A critical aspect of our design model involves the requirement by the designer to specify the *method profile* for each method of an application:

Definition 4 : Each method $M_i \in \mathcal{M}$ is defined using a *method profile*, MP , that contains:

1. a prose *method description* for the method's actions within the application.
2. the method's name, its return type, a list of parameters (with their names and types), and the portion of private data of the OT that the parameter maps to.
3. The read/write set for each private attribute of the OT that is used by the method M_i , called \mathcal{RW}_i .
4. The other methods that are called by M_i to accomplish its task, referred to as \mathcal{OM}_i .

For example, the method profile of a push operation for a stack OT contains its description (*Push modifies the stack by inserting a new top element as long as the stack has space remaining.*), its parameters ($e:char; s:stack$), its return type ($stack$), a mapping of the parameters to private data (e to top and s to st , where top and st are private data items), the read/write set (top/w and st/w), and the other methods called ($is_full()$).

There are many possible interpretations of inheritance, differing in the data and methods of the supertype that are available for use by the subtype. Our version of inheritance is:

Definition 5 : $OT_2 \text{ ISA } OT_1 \Rightarrow OT_2 = (OTName_2, \{D_2\}, \{M_2\})$ with PPI methods from the supertype only available via security authorization.

OT_2 as a subtype of OT_1 permits OT_2 to be used when an OT_1 instance is expected (substitutability). If OT_2 needs access to OT_1 's PPI methods, they must be explicitly authorized.

Other relationships in addition to inheritance are also possible:

Definition 6a: $OT_1 \text{ REL } OT_2 \Rightarrow OT_1$ can call the PPI methods of OT_2 . REL has an associated set of PPI methods to support the behavior of the relationship.

Definition 6b: $COL(OT_1)$, short for COLlection, is a relationship involving a single OT, can call the PPI methods of OT_1 , and includes a set of PPI methods for managing the instances of a collection.

For example, if REL was 1-to-many, the PPI methods would support the addition and deletion of source and destination instances. A different relationship (say, is-part-of), would have different PPI methods. SET is similar to a COL relationship, with duplicates not allowed.

2.2 The Health Care Application

In Figure 2, we present a probable characterization of the main object types and their associations via inheritance that are appropriate for a health care database. We have based this characterization on discussions and input from those familiar with the health care profession (see Acknowledgement). We do not claim that this characterization is complete, but we do believe it gives strong indications on the different and diverse database needs and requirements for health care information.

The major OTs are Person, Item, Formulary, Record, Organization, and Budget, as indicated in Figure 2a. Person is expanded in Figure 2b to contain Patient, Physician, Staff, and Administrator, where Staff has many different subtypes. Each subtype of Staff represents many different job categories, e.g., Nurse represents RNs, LPNs, and Nurse's Aides, Support includes Housekeepers, Security, Electricians, Carpenters, etc. Similar enumerations are also present in other parts of the hierarchy, such as the specialties of physicians, different Radiology Tests (X-ray, CAT scan, MRI, etc.), and so on. Figure 2a also indicates that for each patient, a Medical Record is maintained that contains a collection of Items. Visit, Prescription, and Test are all subtypes of Item, and indicate procedures performed on a patient's visit to a physician. Each Visit can be further

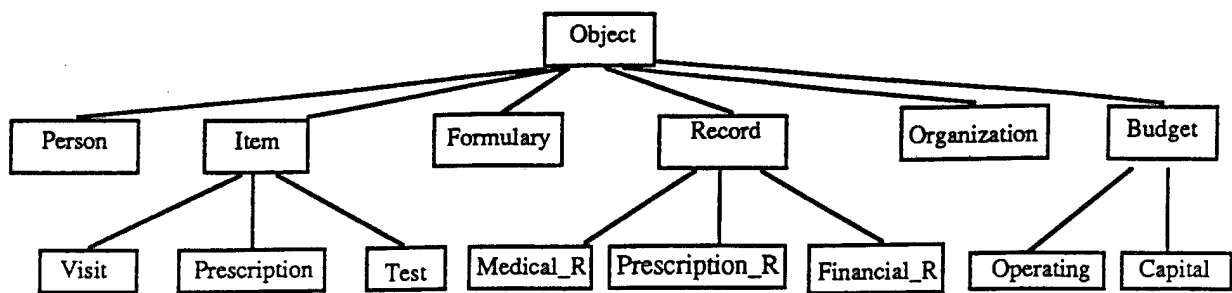


Figure 2a: The Major Object Types.

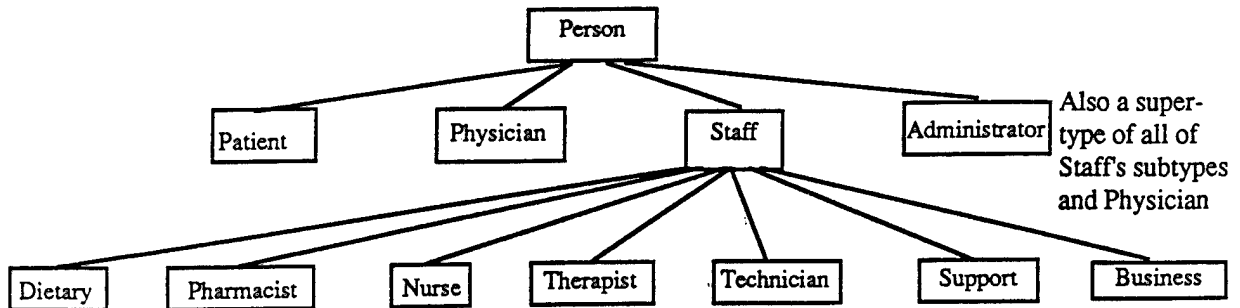


Figure 2b: The Person Object Type and its Subtypes.

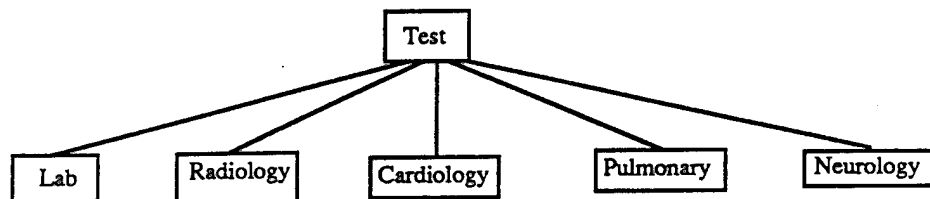


Figure 2c: The Test Object Type and its Subtypes.

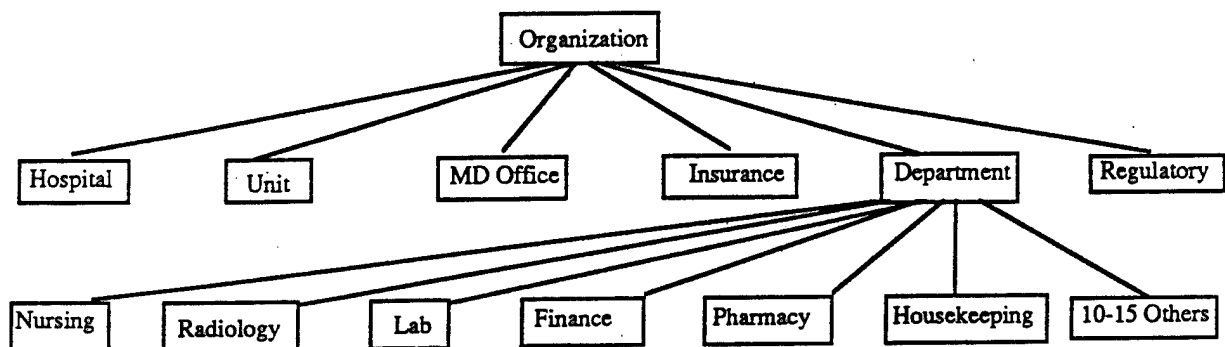


Figure 2d: The Organization Object Type and its Subtypes.

Figure 2: A Probable Conceptualization of Health Care Data with Inheritance.

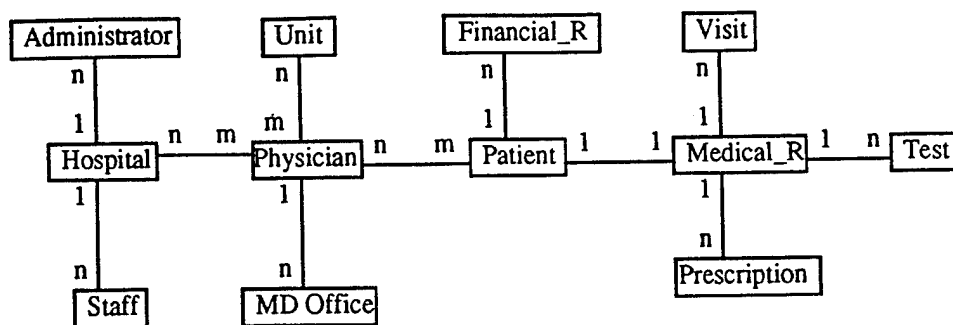


Figure 3: A Subset of the Possible Relationships.

specialized using inheritance with OTs for Inpatient and Outpatient visits (not shown in Figure 2). The Test OT has many subtypes that are distinguished functionally.

Elsewhere in Figure 2, Records are maintained on all prescriptions (Prescription_R) filled by the Pharmacy (for accounting purposes) and for all revenue generating requirements (Financial_R). The Formulary OT refers to an entire sub-hierarchy for a database of all drugs and their interactions, which pharmacists utilize to notify physicians when there is a conflict between the drugs prescribed for a patient. The other major portion of the figure, is the Organization OT and all of its subtypes, shown in part in Figure 2d. There are many different organizations, including the Hospitals, Units, MD Offices, Insurance agencies, Departments, and Regulatory agencies (oversee and accredit health care). Departments are the major divisions of responsibility within a hospital. We have listed six major departments (Nursing, Radiology, Lab, Finance, Pharmacy, and Housekeeping) and have omitted other smaller ones. The functional Units (previously called wards) within a hospital are quite broad, and include: Critical Care, Rehabilitation, Surgical, Medical, Recovery Room, Emergency Room, Maternity, Operating Room, Psychiatry, Outpatient, Neurology, and Nursery. These Units correspond to where the direct care is administered to patients.

To complement the OTs given in Figure 2, we have developed a partial subset of the relationships between OTs, as given in Figure 3. In the figure, we have established associations between the different OTs. For example, a Hospital contains many Administrators and Staff, and has a number of Physicians in various capacities (e.g., attending, residents, interns, administrators, etc.). Each Physician can have privileges at many Hospitals, can see one or more Patients on many Units, and may have one or more Offices at various locations. Also, for a Patient, many Physicians may be consulted, and multiple Financial_R(ecords) (for each visit) and a single Medical_R(ecord) (for all visits) are maintained. A Medical_R(ecord) contains the Visits, Prescriptions, and Tests that catalog the complete medical history of the Patient.

Finally, in Figure 4, we include OT descriptions for Item, Visit, Prescription, Test, Record, and Medical_R. We have omitted the Prescription_R and Financial_R subtypes of Record shown in Figure 2a from Figure 4. For each of these OTs, we have indicated both the private data and the PPI methods. Much of the information in this figure has

been motivated and developed from examples in [31]. We have included a wide range of basic methods that create and retrieve the different information. For clarity, each of the methods is numbered with MX.Y. The X refers to an OT. The Y refers to a method within an OT. We have omitted the method profile for each method. Information from the method profile will be provided as needed in Section 3.

Finally, we must introduce one last concept related to Definition 5 in Section 2.1. First, we define the *local public interface*, or *LPI* of the OT to be the PPI at individual nodes in an application, without considering inheritance. In Figure 4, the LPI of Visit contains the methods M2.1 through M2.6. Second, we define the *global public interface*, or *GPI*, to be the set of all LPIs from the current node through its ancestors from inheritance. Referring to the example again, the GPI for Medical_R would contain M6.1 through M6.12 (its own LPI methods) and M5.1 to M5.6 (LPI methods from Record). Third, the *aggregate public interface*, *API*, is the set of all possible potential public methods, namely, the union of all methods in all LPIs regardless of type boundaries, i.e., all methods MX.*, where X ranges over all OTs of an application. Fourth, the *unified public interface*, *UPI*, is {LPI., GPI., API}, the set of all possible interfaces. The different interfaces are easily and automatically constructed and maintained while an application is being designed.

3 Defining User-Role Based Security

Once the OTs of an application have been developed (see Section 2.2 and Figures 2 and 4), the security designer can begin the process of defining the user roles and their associated security privileges. This section describes this process by: reviewing the concept of a user-role definition hierarchy for organizing user roles; introducing the concept of a node profile which contains descriptions of the role responsibilities, the methods to be assigned and prohibited for each role, and consistency criteria for roles; and, presenting new analysis techniques which assist the security designer in establishing precise and correct privileges for their application. Throughout the section, we elaborate on a specification methodology for supporting the definition of user roles and the analyses of their privileges.

3.1 The User-Role Definition Hierarchy

A *user-role definition hierarchy (URDH)*, is used by the security designer to characterize the different kinds of individuals (and groups) who all require different levels of access to an application. We employ a hierarchy to represent associations between individuals (or groups) with different, yet related, needs. We characterize the responsibilities of individuals into three distinct levels of abstraction for the URDH: user roles, user types, and user classes. *User roles* allow the security designer to assign particular privileges to individual roles. For example, user roles for the registrar's office of a university might be *grade-recorder* or *transcript-issuer*. These two roles are different; a *grade-recorder* enters changes and checks corrections; a *transcript-issuer* will have different access.

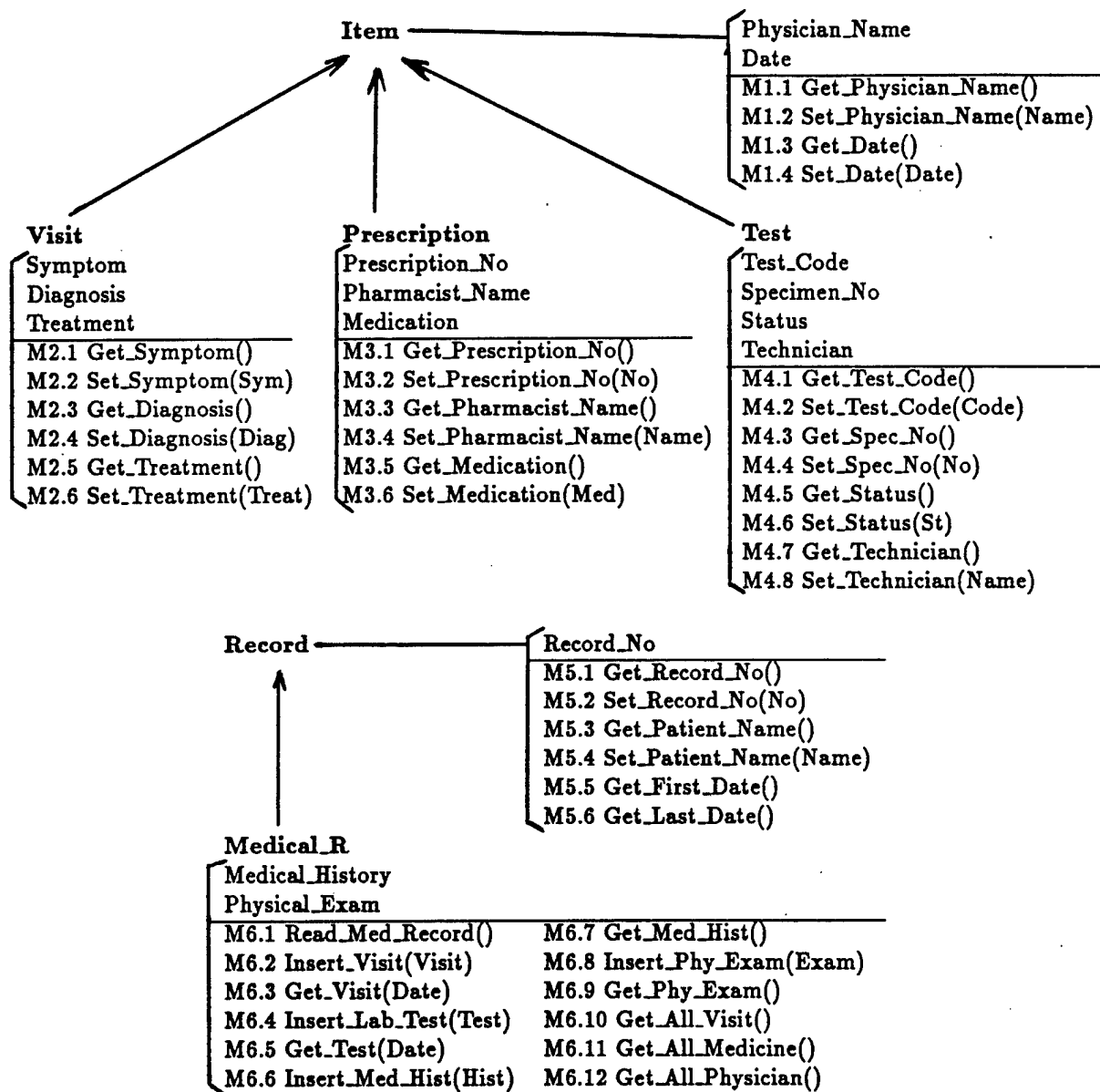


Figure 4: Selected Object Types and their Private Data/PPI Methods.

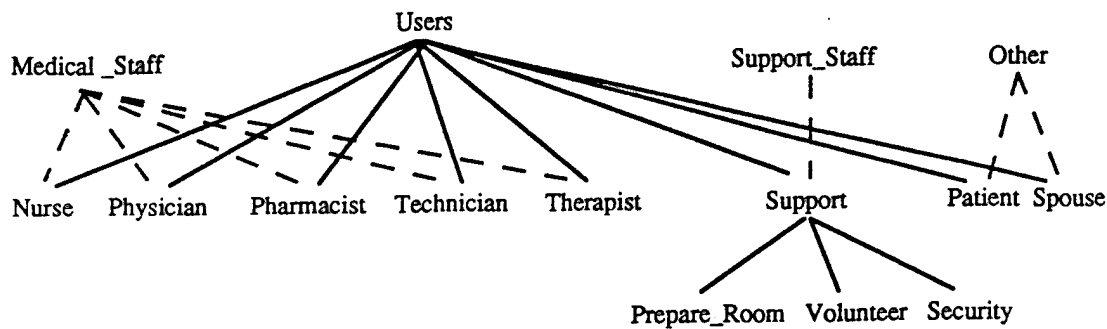


Figure 5a: User Types, User Classes, and Selected User Roles.

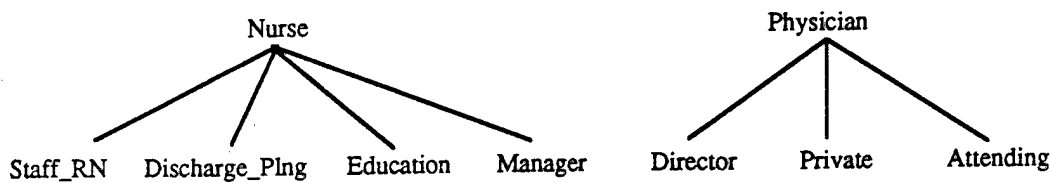


Figure 5b: User Roles for Nurse and Physician.

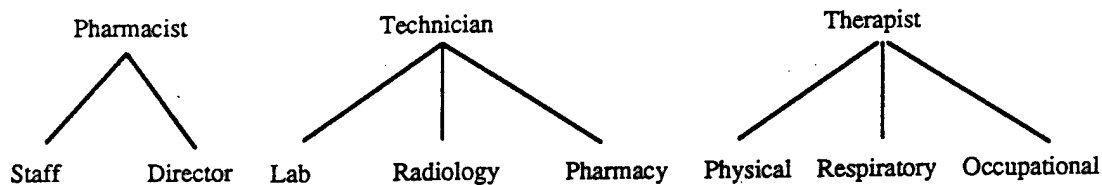


Figure 5c: User Roles for Pharmacist, Technician, and Therapist.

Figure 5: User Roles, User Types, and User Classes for Health Care.

To represent similarities that exist among user roles, a *user type* can be defined. For example, the user roles grade-recorder and transcript-issuer can be grouped under the user type registrar-staff. User types characterize common responsibilities among related user roles, e.g., all registrar-staff access names and addresses in student records. Privileges that are assigned to a user type are systematically passed to all of its user roles. The different user types of an application can be grouped into one or more *user classes*. For a university, appropriate user classes might be non-academic-staff (with types such as purchasing-staff, campus-police, maintenance-staff, etc.) and academic-staff (with types such as dept-staff, registrar-staff, presidents-office, etc.). The responsibilities are broadly classified into two user classes. Privileges that are supplied to each class are passed on to its types and their roles. The grouping of user types into user classes is very application-dependent. Finally, note that our URDH is analogous to the role lattice proposed in [20,21] and the role network discussed in [18].

We further illustrate the aforementioned concepts with a partial URDH for the health care application in Figure 5. In the figure, the roles are defined in a two-step process

of specialization (top-down definition) and generalization (bottom-up definition). From a top-down perspective in Figure 5a, there are eight different user types: Nurse, Physician, Pharmacist, Technician, Therapist, Support, Patient, and Spouse. In this case, the security designer is assuming that each of these user types may have privileges that would be common to all user roles under the type. Within each user type, one or more user roles may be defined. For example, in Figure 5b, user roles for Nurse include Staff_RN, Discharge_Plng (planning), Education, and Manager. In this top-down examination, *specialization* identifies the various users and their roles. Figure 5a can also be examined from a bottom-up perspective to determine the common characteristics by the grouping of the user types into user classes Medical_Staff, Support_Staff, and Other. This is one possible grouping of user types. The bottom-up perspective corresponds to *generalization* for establishing the user classes, as shown by the dashed lines.

3.2 The Concept of a Node Profile

To more accurately characterize the capabilities of user roles in the URDH, with respect to the privileges to be granted against the application, we propose the creation of a *node profile*. A node profile contains the assigned methods (the privileges for each URDH node), as we have described in our earlier work [6,30]. A node profile also includes a node description, the prohibited methods, and consistency criteria.

3.2.1 Node Descriptions

Node descriptions are utilized to specify the responsibilities of a URDH node via a concise prose statement. Below, are node descriptions for our example of Figure 5:

Medical_Staff: *Collectively, responsible for all aspects of direct patient care.*

Support_Staff: *Different support personnel that address non-medical needs of patients and maintain the physical building.*

Other: *Other individuals that have the potential to access limited portions of the health care database.*

Nurse: *Direct involvement with patient care on a daily basis.*

Physician: *Handle the medical needs (diagnosis, treatment, etc.) for patients.*

Pharmacist: *Control the supply and distribution of all drugs throughout the hospital.*

Technician: *Provide a variety of medical testing support for Patients.*

Therapist: *Evaluate patients and develop treatment plans for therapy.*

Staff_RN: *Administer direct care to patients and implement the physician treatment plan.*

Discharge_Plng: *Link between patients and outside agencies for care after discharge.*

Education: *Educate both the nursing staff and patients regarding new treatments and self care.*

Manager: *Responsible for the day-to-day operation of a nursing unit.*

Director: *(For Physician or Pharmacist) Responsible for the day-to-day operation of their respective department/medical service.*

Private: *The physician within his/her office/private-practice setting.*

Attending: *A physician that has privileges to admit and treat patients at a hospital.*

Staff: *Responsible for filling prescription orders for patients and analyzing appropriateness of drugs and dosages.*

Lab: *Perform/collect different tests involving body/blood on patients.*

Radiology: *Perform radiology based tests/treatments on patients.*

Pharmacy: *Distribute drugs to specific patients at correct times.*

Physical: *Perform physical therapy on patients at prescribed times.*

Respiratory: *Perform respiratory therapy on patients at prescribed times.*

Occupational: *Perform therapy geared towards returning the patient to the independent activities of daily living.*

Support: *Limited contact with patients on a day-to-day basis.*

Prepare Room: *Clean and prepare room after a patient is discharged.*

Volunteer: *Satisfy the needs and interests of patients by offering activities, reading materials, etc.*

Security: *Involved when prisoners/VIPs must be guarded/protected.*

Each of these descriptions can be supplied as the security designer is creating the URDH and its roles, types, and classes, and may be refined/modified as needed.

3.2.2 Assigned Methods

Once the URDH has been specified, the security designer assigns methods to hierarchy nodes, to characterize the privileges based on his/her understanding of the security requirements. We have focused on the methods that are assigned from the application (see Figure 4), with the data and its access intentionally obscured [6,30]. In [20,21], implication rules compute implicit authorization which is similar to our process of method assignment, but differs since they assign objects and authorization types to roles. Our approach also contrasts to [18], where the access rights/permitted roles are assigned based on data levels. Finally, our approach and [23] both have the goal of providing different interfaces to different users, but differ since they assign views based on data.

To support the method assignment process, the information required by each user and the associated privileges (e.g., read, write, or both) must be understood. This corresponds to the application's intended security requirements, and is developed by the security designer as part of the requirements definition for the application under development. We use the term access to mean read only. Explicit write needs are noted separately. Also, the term *clinical information* is broad, and coincides to Medical Records, Visits, Prescriptions, and Tests (all information on a patient). Below is a list of the information accessing requirements for our health care example:

Staff_RN: All clinical information for the patients that they are responsible for (referred to subsequently as clinical info.). Can write/modify a substantial portion of clinical information to record the results/patient progress. Cannot change a Physician's orders on a patient.

Discharge_Plng: All clinical info. for patients. In addition, financial information is consulted, since patients might be placed in a continuing care facility or may require home visits from various health care professionals. Don't have as much write access to clinical info. of a patient as Staff_RN, but can write notes. Cannot change a Physician's orders on a patient.

Education: More limited access to clinical data than Staff_RN, but since they do teach patients after-discharge care (e.g., diabetic care, etc.), they do need access to a patient's history. Like Discharge_Plng, can write notes which document a patient's progress. Cannot change a Physician's orders on a patient.

Manager: All clinical info. plus information required to transfer patients between units, information on the nurses that work in their unit (including shifts, staffing, and skill levels), and budgetary data. Write privileges of Staff_RN plus extra privileges to read information on other units (censuses) and write summary and employee data on their own units. Cannot change a Physician's orders on a patient.

Director: (For Physician) Information on physicians in their departments, budgetary data, clinical information on patients including summary data on trends and volumes. Write ability on employee data.

Private: All clinical info. on patients and who they can contact (nurses, technicians, therapists, etc.) regarding their patients. Also, insurance related data and other office-based data to maintain their private practice. Overlap of write privileges with Staff_RN, but can also modify portions of clinical info. that issue orders.

Attending: All clinical info. on their patients. Similar to Private.

Director: (For Pharmacy) Information on pharmacists and technicians that are employed, budgetary data, summary information on drug distribution and usage, limited clinical info. on patients. Write ability on employee data.

Staff: Access to the Formulary database, all clinical info. on patients due to possible drug interactions, and prescription records.

Lab: Limited access to clinical info. on patients. They need to know what tests are required for which patients and when they are to be performed. Limited write access on clinical info. to record test results of patients.

Radiology: Similar to Lab user role.

Pharmacy: Similar to Lab user role. May need access to Formulary database.

Physical: Access to clinical info. that is greater than Lab user role but less than Staff_RN. Can write notes on patient's progress which are permanently recorded into the medical record.

Respiratory: Similar to Physical user role.

Occupational: Similar to Physical user role.

Prepare_Room: *Very limited clinical info. on patients - discharge date and time. No write access is allowed.*

Volunteer: *Very limited clinical info. on patients - names, location, restrictions (food/smoking), and interests. No write access is allowed.*

Security: *Very limited clinical info. on patients - duration/location of prisoners/VIPs. No write access is allowed.*

Using this information, the security designer can establish privileges for URDH nodes.

To illustrate the method assignment process, we highlight some possible assignments for the URDH in Figure 5 against the health care database of Figure 4. First, consider the user roles of Nurse: Staff_RN, Discharge_Plng, Education, and Manager. Individually, they all need read/write access to clinical information on patients. Staff_RN would likely have access to most Get methods from Figure 4 (14 methods), including, for example, Get_Symptom of Visit, Get_Medication of Prescription, Get_Patient_Name of Record, and Get_Test of Medical_R, but may be unable to access the Get_All methods. Staff_RN would also access the two read methods and a selected subset of the Set/Insert methods, e.g., Set_Symptom (record symptoms on patients), Set_Test_Code (record test to be conducted), Set_Patient_Name, Insert_Visit, Insert_Med_History (taken by nurses), etc. Discharge_Plng and Education nurses would have similar Get access, but more restricted write access, say Insert_Visit or Insert_Med_History, since the information accessing requirements state that they only write notes on patient progress. Managers would have all of the access of Staff_RN, but may also have additional access to fulfill their role (such as the Get_All_Medicine and Get_All_Physician methods).

Methods are also assigned using the different public interfaces that were discussed in Section 2.2. For example, the user type Nurse could be assigned LPI of Record, since all of its roles can access all of these methods. The role Physician/Private could be assigned LPI of Medical_R, indicating that the doctors are able to invoke all of the methods which are defined (M6.1 to M6.12). A better assignment for this role would be the GPI of Medical_R, since the GPI also includes all methods from Record, which doctors should also be able to utilize, e.g., doctors need access to the patient's name and visit history that is maintained in Record.

From the above discussion, a methodology for method assignment begins to appear. All privileges are assigned at the bottom-most level of the URDH, corresponding to the user roles. Given the privileges assigned to roles, the security designer can examine the roles under a single user type (in this case, Nurse), and seek to identify commonalities between the specific assignments. Commonalities with respect to shared privileges are pushed up the URDH from the user roles to their shared user type. In this case, all of the Get methods and any common Insert methods can be moved up to the Nurse type. When multiple user types have the same user class (e.g., Nurse and Physician are under Medical_Staff in Figure 5), common methods may also be moved up to the user class. However, the methods must be common to all user types of a specific user class, or a type/role may acquire methods on which privileges were not intended. For example, all user types of Medical_Staff would have access to the Get_Patient_Name method of Record. A method that is shared by all user types of a URDH can be moved

up to Users. `Get_Patient_Name` may be such a method for this application. Thus, the specification methodology indicates that method assignments at the user roles flow up the URDH to the user types, user classes, and User. Flow of common information up the tree effectively forces differences in method assignments to be pushed down the tree.

3.2.3 Prohibited Methods

The *prohibited methods* on a URDH node represent those methods which cannot be accessed by the URDH node. Thus, the security designer augments the positive actions of the node (i.e., the assigned methods) with the non-allowed actions (i.e., the prohibited methods), as reflected in the information access requirements of the user roles (see Section 3.2.2). Our concept of prohibited methods is similar to the concept of denied roles in [18] and permission tags in [3], but differs since both of their efforts emphasize data; we focus on types/methods.

Prohibited methods are very important in the overall specification of privileges. Recall from Definition 4 in Section 2.1, that a method profile contains, the methods which a method calls. Thus, there is the potential to call a great number of different methods which are defined on many different object types and that involve numerous private data items, when a single method is assigned to a URDH node. Since this is the case, the prohibited methods can be utilized to explicitly identify which methods cannot be accessed by a URDH node. With this information, the analyses we will discuss in Section 3.3 can automatically inform the security designer when a prohibited method conflicts with a assigned method (or a method called by a assigned method, and so on) on a URDH node.

In our example, the security designer can explicitly list the methods that the different user roles cannot access. For different roles of Nurse: `Staff_RN` cannot access `Set_Treatment of Visit`, `Set_Medication of Prescription`, `Get_All` methods from `Medical_R`, and so on; `Discharge_Plng` and `Education` would have a larger exclusionary list; and `Manager` would be similar to `Staff_RN`, but may be able to utilize some of the `Get_All` methods that `Staff_RN` cannot. A similar specification methodology to the steps described for assigned methods is also employed for prohibited methods. Common prohibited methods will pass up the URDH via the paths from user roles to a user type, and from user types to a user class. However, in this case, the prohibited methods are not explicitly repositioned in the URDH. Rather, the semantics of prohibited methods imply that a method prohibited from a user role (type), is also prohibited from its associated user type (class).

3.2.4 Consistency Criteria

Consistency criteria in a node profile relate any two user roles, types, or classes with respect to their capabilities. *Equivalence criteria* allow the security designer to identify which user roles (types/classes) must have the same capabilities, as reflected in the methods, OTs, private data that are assigned/prohibited. Equivalence criteria are very important for defining URBS, since whenever a change is made to the URDH (assigned/prohibited method is added/removed), the security designer can be alerted that

the privileges are no longer equivalent and particular nodes must be also modified. For example, from the information accessing requirements in Section 3.2.2, the roles Physical, Respiratory, and Occupational of the Therapist user type, would all be equivalent, and a change to one role, would require a corresponding change to the other two.

Subsumption criteria allow the security designer to establish an ordering among URDH nodes, indicating that the capabilities of one node cannot exceed the capabilities of another node. In our example, there are many subsumptions that should be specified by the security designer. Both Education and Discharge_Plng are subsumed by Staff_RN (since the latter writes more portions of the database) which is subsumed by Manager. Education is also subsumed by Discharge_Plng, since the latter requires access to financial information. Physical, Respiratory, and Occupational roles are subsumed by Education, since the former three have limited database access. The three types of Therapist are also subsumed by Discharge_Plng, Staff_RN, and Manager, via transitive closure. Subsumptions at the user type level are also possible, e.g., Technician subsumed by Therapist. A formal definition of consistency criteria is:

Definition 6: Let O be the set of OTs that can be accessed by a URDH node N , M be the set of methods that can be assigned and/or prohibited by N , and P be the set of the private data that can be accessed by N . Let equivalence be represented by \equiv , and subsumption be represented by \triangleleft . Then, we define $N_i \equiv (\triangleleft)N_j$ with respect to OTs if $O_i = (C)O_j$, $N_i \equiv (\triangleleft)N_j$ with respect to methods if $M_i = (C)M_j$, and $N_i \equiv (\triangleleft)N_j$ with respect to private data if $P_i = (C)P_j$.

The different criteria are checked by comparing methods, OTs, or private data.

3.3 Analyses for Design Feedback

To assist the security designer in the definition of privileges on the URDH via the node profiles, a set of new analysis techniques are provided. Some of these techniques are designer initiated; others automatically alert the designer to possible conflicts and inconsistencies. Designer-initiated analyses on a chosen URDH node are: a summary of the node descriptions including ancestors; a summary of the methods which have been assigned/prohibited (including ones acquired from ancestors); and, a summary of the method descriptions for assigned methods. Automatic analysis techniques are: the identification of conflicts between assigned/prohibited methods as reflected in the methods, OTs, or private data items of the application; checking the consistency of the different equivalences and subsumptions; and, alerting the designer when privileges given/removed to a chosen URDH node must also be made to other nodes to maintain existing equivalences or subsumptions. Also, since method calls can be nested, all of the analyses can be performed recursively to any desired level of depth.

3.3.1 Node-Description Summary

From a strictly informational perspective, the security designer can aggregate or summarize the node descriptions for a chosen URDH node to check whether the written

descriptions correspond to their interpretation of the node's responsibilities, and to update the node profiles if necessary. In our example, when the designer chooses Staff_RN, the following descriptions are supplied:

Medical_Staff: *Collectively, responsible for all aspects of direct patient care.*

Nurse: *Direct involvement with patient care on a daily basis.*

Staff_RN: *Administer direct care to patients and implement the physician treatment plan.*

If Support is chosen, then the descriptions:

Support_Staff: *Different support personnel that address non-medical needs of patients and maintain the physical building.*

Support: *Limited contact with patients on a day-to-day basis.*

are returned, indicating that roles under this type interact minimally with patients. Node-description summaries are most likely performed as the designer is creating the URDH, by defining roles, types, and classes. The security designer is using the summaries to verify whether the hierarchy has been correctly structured as reflected in the aggregated descriptions.

3.3.2 Capabilities Analyses

Capabilities analyses allow the security designer to review the permissions given to a chosen URDH node on an application's OTs, methods, and private data. This review can occur throughout the time period when the designer is defining the URDH and establishing assigned/prohibited methods for its nodes. For example, the designer can choose the Staff_RN node and be presented with the following information:

- all methods which have been assigned to Staff_RN and its ancestors (Nurse, Medical_Staff and Users);
- all OTs which can be accessed by Staff_RN, since each assigned method belongs uniquely to a single OT; and
- all private data which is accessed by Staff_RN, since each assigned method uses private data in a read, write, or read/write fashion.

This information corresponds to a first-level inspection of the implications of the method assignments of Staff_RN (and its ancestors). Analysis is also available in a similar fashion for the prohibited methods and the OTs and private data which cannot be accessed. These analyses are supported at a direct level as described or an indirect level (since methods calls can be nested). Algorithms for the assigned method analyses appear elsewhere [6]. The Appendix contains the direct techniques for capabilities analyses of prohibited access.

3.3.3 Method-Description Summary

Once the method-assignment process begins, the security designer can initiate method-description summary analysis, to aggregate the functional descriptions of the methods assigned to a chosen URDH node. For example, if the role Technician/Lab is selected, the following method descriptions would be provided:

M5.3 Get_Patient_Name(): *Identify the patient by name.*

M5.1 Get_Record_No(): *Find a specific medical record by number.*

M4.8 Set_Technician(Name): *Set the name of the technician that performs the test.*

M4.2 Set_Test_Code(Code): *Set the code of the test to be conducted.*

M4.4 Set_Spec_No(No): *Set the specimen number for the test.*

M4.6 Set_Status(St): *Set the status for the test.*

M6.4 Insert_Lab_Test(Test): *Insert a lab test result.*

These descriptions are from the URDH, e.g., M5.3 could be assigned to Users, M5.1 to Medical.Staff, M4.8 to Technician, and M4.2, M4.4, M4.6, and M6.4 to the Lab role. These assignments are consistent with the node description for Lab (i.e., *Perform/collect different tests involving body/blood on patients*) and its information accessing requirements in Section 3.2.2 (i.e., *Limited access to clinical info. on patients. They need to know what tests are required for which patients and when they are to be performed. Limited write access on clinical info. to record test results of patients*).

3.3.4 Conflict Identification Analyses

When the privileges which have been granted to a URDH node are in conflict, the security designer must take action to resolve the identified problem. *Conflict identification* is automatically performed whenever changes are made to a URDH node via assigned/prohibited methods. As a result of the identification, the security designer updates the assigned and/or prohibited methods to correct the problem. Three automatic analysis techniques are supported for conflict identification, based on methods, OTs, and private data. As the security designer is assigning/prohibiting methods to URDH nodes, s(he) is receiving real-time feedback when a method that causes a conflict (in a method, OT, or private data item) is inserted into a node profile. Like the capabilities case, both direct and indirect analyses are supported.

To illustrate conflict identification analysis, consider the example from the previous section for the Technician/Lab role, and add the prohibited methods Get_Test_Code, Get_Spec_No, Get_Status, and Get_Technician from the Test OT to the node profile of this role. Now consider that at some later point in time, the security designer attempts to add the method Get_Test(Date) of Medical.R to the Lab role, and suppose that Get_Test calls the Get methods from Test to return the information on a lab test. When this assignment is attempted, since Get_Test calls methods that are prohibited for the Lab role, the designer would be notified that their intended assignment cannot

be made. Other conflicts can occur when a prohibited method is assigned to a chosen URDH node that conflicts with an existing assigned method (e.g., the prohibited method calls methods that have already been assigned to the node), and may also be evaluated from the perspective of OTs and private data.

3.3.5 Consistency Criteria Checking

A second analysis technique would automatically check the consistency criteria between URDH nodes in two ways: as equivalences and subsumptions are being defined among URDH nodes and whenever changes in the form of additions/deletions of assigned/prohibited methods are made. Whenever the designer is alerted that an inconsistency to a subsumption or equivalence has occurred, modifications of privileges to correct the problem must be made. To illustrate the first checking, suppose that the following subsumptions have been established:

Education \triangleleft Discharge_Plng \triangleleft Staff_RNs \triangleleft Manager

If, at some later time, the designer attempts to establish either of the two equivalences:

Education \equiv Staff_RNs or Staff_RNs \equiv Manager

s(he) would automatically be informed of inconsistencies with existing criteria. These checks are especially useful when the criteria involve user roles that are specialized from different user types.

In the second checking, criteria are examined for consistency with respect to the application's methods, OTs, and private data as reflected in the assigned/prohibited methods for the URDH. Both equivalences and subsumptions are checked for methods, OTs, and private data with respect to assigned and prohibited methods (a total of 12 checks). Checking is also supported at direct and indirect levels (as in the capabilities and the conflict identification cases), resulting in a total of 24 analysis techniques for verifying consistency. To illustrate this checking, again consider the subsumptions for Nurse roles shown above, with Manager assigned Get_All_Medicine and Get_All_Physician of Medical_R, but not Get_All_Visit. If at some point during the definition of privileges, the security designer assigns Get_All_Visit to Staff_RN, s(he) would be automatically informed that such an assignment would violate an existing subsumption. Subsumptions and equivalences can also be violated if one URDH node can access an OT (or private data item) that the other cannot, e.g., Education can access a Nutrition OT that Discharge_Plng cannot.

To support consistency checking during the security-definition process, we allow the designer to establish a set of default consistency checks that s(he) wishes to be automatically verified as the URDH is being developed, customizing the checks to suit a designer's needs. For example, a designer may want the checks to only occur automatically for assigned and prohibited methods, and not for OTs or private data items. All checks which are not automatically provided can be explicitly initiated by the designer.

4 Methodological Issues and Considerations

This section discusses issues related to the specification methodology. We begin by reviewing the steps of the methodology. We then provide a post-mortem on the applicability of the methodology for the health care example. Finally, we discuss other experiences that have impacted on the methodology.

4.1 A Specification Methodology

Throughout Section 3, when examining the definition and analyses available for developing the user roles and their privileges, we have also emphasized a methodology for specifying this information in a controlled fashion. In this section, we synopsise and justify the features of the methodology. There are four major steps in the methodology. The first step corresponds to the development of the URDH by the security designer, and can be subdivided as:

- 1a. Define user types and user roles in a top-down fashion with specialization.
- 1b. Define user classes in a bottom-up manner with generalization.
- 1c. Coincident with these two definition steps is supplying node descriptions and invoking summary analysis (Section 3.3.1) on nodes to verify the aggregate descriptions.

With these actions, the security designer obtains an initial characterization of the URDH.

Once the overall structure of the URDH has stabilized, the security designer can proceed to assign methods to URDH node, which corresponds to the positive actions or capabilities. In this process, the designer is using the information accessing requirements (Section 3.2.2) as a guide for insuring that the correct privileges are given. There are five substeps:

- 2a. Assign methods starting with the user roles under a user type.
- 2b. Move methods shared by all user roles to its common user type.
- 2c. Move methods shared by all user types to its common user class.
- 2d. Move methods shared by all user types to Users.
- 2e. Perform capabilities (Section 3.3.2) and method-description summary (Section 3.3.3) analyses after any of the above steps to verify that assigned methods correspond to intent.

The result of this step in the methodology is an initial identification of the privileges for each URDH node.

The third step in the methodology focuses on actions which are to be explicitly prohibited from access by the URDH nodes. This step is divided into:

- 3a. Associate prohibited methods with user roles (types), which implies the methods are also prohibited on the roles shared user type/class (class).

- 3b. Perform capabilities analyses (Section 3.3.2) for prohibited methods to insure that the correct non-allowed actions have been specified.

Once the security designer starts to add prohibited methods to the node profiles, conflict identification analyses (Section 3.3.4) automatically begins, to notify the designer when assigned/prohibited methods contradict.

The fourth and final step of the methodology is used by the security designer to specify consistency criteria among the URDH nodes:

- 4a. Define equivalence criteria between any two user roles, types, or classes.
- 4b. Define subsumption criteria between any two user roles, types, or classes.

As criteria are being defined, automatic analyses verify that the criteria do not conflict with one another (first check of Section 3.3.5). Analyses also alert the designer when privileges that are assigned/prohibited to a URDH node violate the existing consistency criteria (second check of Section 3.3.5). The actions taken in steps 1, 2, 3, and 4 of the methodology are iterative, incremental, and cyclical. They can be repeatedly performed by the security designer until s(he) is satisfied that an accurate characterization of user roles and their capabilities have been developed.

4.2 The Applicability of the Methodology

The specification methodology has adapted well to the health care case, and while our example was not complete, it was complex enough to illustrate both the applicability and utility of the methodology. The definition of the URDH can represent traditional roles such as Staff_RN, Director, Therapist, etc., quite well. Assigned/prohibited methods allow the positive and non-allowed actions to be clearly identified. Subsumption and equivalence can be employed to establish associations/invariants that must exist among URDH nodes. The various analysis techniques (Section 3.3) complement the definition process, and provide tools for the security designer to arrive at a more precise and correct characterization of roles and privileges.

However, we have yet to explore the user roles of patient and spouse (see Figure 2), which we believe will be the most difficult with respect to understanding their responsibilities and capabilities. Also, the concept of a sub-role of a user role needs investigation, though this may be achieved in our approach by simply defining finer grained user roles. Another glaring absence in the methodology is a lack of integrity and security constraints. Integrity constraints are required to control and maintain data consistency. Content and context based security constraints are necessary to allow the privileges to be more specialized (e.g., a Staff_RN role can only access the Patients that occupy their Unit). These and other issues are currently being explored to support URBS more completely for real-world applications.

4.3 Other Related Experiences

Finally, we mention other related experiences that have, in part, driven our research directions towards a specification methodology for URBS. As part of a Ph.D. qualify-

ing examination in January, our baseline URBS approach [6,30] was applied by three students against a university database example. One exam was particularly interesting, and contained the most extensive version of a URDH that has been developed so far based on our approach. The object-oriented design of the university database had a total of 11 OTs, with 54 total methods. The URDH contained 20 user types grouped under three user classes. Each of the 20 user types had an average of two user roles (47 total). The answers to the exam has impacted on our work, especially with respect to the assignment of privileges to URDH nodes in step 2 of the specification methodology.

5 Concluding Remarks and Ongoing Research

In this paper, we have presented a specification methodology for supporting the precise and accurate characterization of user-role based security (URBS) for an object-oriented design model. The core of the methodology is supported by: a user-role definition hierarchy (URDH) for specifying the different user roles, user types, and user classes for an application; node profiles that allow the security designer to define the privileges of each URDH node using a node description, assigned/prohibited methods, and equivalence/subsumption criteria; analysis techniques (designer initiated and automatic feedback) that provide, for a URDH node, summary information (on node and method descriptions), capabilities (with respect to assigned/prohibited methods), conflict identification (between assigned/prohibited access privileges), and consistency checking (for equivalence/subsumption criteria). To illustrate the utility of the methodology we have used a health care application.

Overall, we believe that the following conclusions can be made regarding the specification methodology:

1. The methodology serves as a systematic discipline.
2. The methodology is consistent, and is divided into a set of well-defined steps or stages that interact well with one another.

We believe that these conclusions have been demonstrated using the health care application.

We are developing a prototype to support the work described in this paper and our earlier research [6,7,30], and expect a baseline system to be available within six months. Our prototyping effort extends our earlier work on object-oriented design tools [4,5,10,11,12]. Over the past two years, we have developed the database design tool ADAM, short for Active DATA Model, that allows an application designer to graphically and textually specify their application behavior and semantics using an object-oriented approach. ADAM supports the modeling constructs of object types, inheritance, and relationships, and automatically generates code and code templates (in C++) for an application's design. Our work on ADAM has also explored application specific database design tools through the development of a version of ADAM that supports scenario design [5,12] for a dynamic distributed decisionmaking environment [15]. In this effort, we are currently examining the ability of URBS for characterizing the roles, information privileges, and sharing requirements of human decisionmakers.

Acknowledgements

The authors wish to extend their heartfelt appreciation to Lois R. Demurjian, R.N., M.B.A., Manager, Nursing Information Systems, at St. Francis Hospital and Medical Center, Hartford, Connecticut, for her significant contributions and input to the examples in this paper. Specifically, the information structure of a health care organization (Figure 2), the user roles and responsibilities (Figure 5), the node-description list (Section 3.2.1), and the information accessing requirements (Section 3.2.2).

Appendix. Algorithms for Analysis Methods

This appendix includes the techniques that support the direct analyses discussed in Section 3.3. The indirect analyses are not shown since they are just extensions of the direct case that consider nested method calls to any level of depth. We have employed an object-oriented approach to defining these techniques, as discussed elsewhere [6,30].

Capability Analyses

Capabilities analyses are performed for a chosen URDH node to determine the privileges that have been granted to the node. There are six analysis techniques for the capabilities of assigned methods, for the access of a chosen URDH node to an application's methods, OTs, and private data. These analyses have been described elsewhere [6]. The method headers are given below since they are used in other analysis algorithms.

```
Node::URDH_DIR_M_A() : Return Method_Set;
Node::URDH_DIR_OT_A() : Return App_Object_Type_Set;
Node::URDH_DIR_PD_A() : Return Private_Data_Access_Set;
Node::URDH_IND_M_A(num_levels) : Return Method_Set;
Node::URDH_IND_OT_A(num_levels) : Return App_Object_Type_Set;
Node::URDH_IND_PD_A(num_levels) : Return Private_Data_Access_Set;
```

In addition to assigned method analyses, there is also capabilities analyses for the prohibited methods, with the techniques shown below:

- Return the prohibited method set of a URDH node.

```
Node::URDH_DIR_P_M_A() : Return Method_Set;
    URDH_DIR_P_M_A ← {prohibited_methods()};
End Node::URDH_DIR_P_M_A;

UserType::URDH_DIR_P_M_A() : Return Method_Set;
    URDH_DIR_P_M_A ← {users_node().prohibited_methods(),
                      user_class_node().prohibited_methods(),
                      Node::URDH_DIR_P_M_A()};
End UserType::URDH_DIR_P_M_A;
```

```

UserRole::URDH_DIR_P_M_A() : Return Method_Set;
    URDH_DIR_P_M_A ← {users_node().prohibited_methods(),
                      user_class_node().prohibited_methods(),
                      user_type_node().prohibited_methods(),
                      Node::URDH_DIR_P_M_A()};
End UserRole::URDH_DIR_P_M_A;

```

- Return the prohibited OT set of a URDH node.

```

Node::URDH_DIR_P_OT_A() : Return App_Object_Type_Set;
    Let MS ← URDH_DIR_P_M_A(); -- MS short for method set
    Let OT_RS ← {}; -- RS short for result set
    Iterate Over All Methods Mi in MS;
        Let ot ← Mi.which_OT(); -- return the OT for Mi
        Let OT_RS ← OT_RS ∪ ot;
    End Iteration;
    URDH_DIR_P_OT_A ← OT_RS;
End Node::URDH_DIR_P_OT_A;

```

- Return the prohibited private data set of a URDH node.

```

Node::URDH_DIR_P_PD_A() : Return Private_Data_Access_Set;
    Let MS ← URDH_DIR_P_M_A();
    Let PD_RS ← {{OT1, {}}, {OT2, {}}, ..., {OTn, {}}};
    Iterate Over All Methods Mi in MS;
        Let ot ← Mi.which_OT(); -- return the OT for Mi
        Let RW_Set ← Mi.read_write_set(); -- return the read/write set of
        Mi
        Let PD_RS ← PD_RS.union(ot, Mi.M_Name, RW_Set);
    End Iteration;
    URDH_DIR_P_PD_A ← PD_RS;
End Node::URDH_DIR_P_PD_A;

```

Indirect techniques are also available, employ similar algorithms, and are omitted for brevity.

Conflict Identification Analyses

Three kinds of analyses can be provided for conflict identification based on methods, OTs, and private data, as a result of the assigned/prohibited methods on a chosen URDH node. The direct analysis techniques are listed below:

- Identify conflict between the assigned and the prohibited methods.

```

Node::URDH_DIR_M_CON() : If conflict, return an error message ;
    Let MS ← URDH_DIR_M_A();
    Let P_MS ← URDH_DIR_P_M_A();
    If MS ∩ P_MS ≠ {}
        Then return error message;
    End Node::URDH_DIR_M_CON;

```

- Identify conflict between the OTs that can and can't be accessed.

```

Node::URDH_DIR_OT_CON() : If conflict, return an error message ;
    Let OT_Set ← URDH_DIR_OT_A();
    Let P_OT_Set ← URDH_DIR_P_OT_A();
    If OT_Set  $\cap$  P_OT_Set  $\neq$  {}
        Then return error message;
End Node::URDH_DIR_OT_CON;

```

- Identify conflict between the private data items that can and can't be accessed.

```

Node::URDH_DIR_PD_CON() : If conflict, return an error message ;
    Let PD_Set ← URDH_DIR_PD_A();
    Let P_PD_Set ← URDH_DIR_P_PD_A();
    If PD_Set  $\cap$  P_PD_Set  $\neq$  {}
        Then return error message;
End Node::URDH_DIR_PD_CON;

```

Indirect analysis algorithms for each of these techniques have also been developed.

Consistency Criteria Checking

Consistency checking is based on an application's methods, OTs, and private data, and can be performed from the perspective of both assigned and prohibited methods (a total of 6 checks). Consistency is also considered for both equivalence and subsumption checks (12 total checks). Similar to previous techniques, direct and indirect analyses are available, resulting in a total of 24 algorithms. The direct analysis techniques for assigned methods are listed below:

- Checks equivalences/subsumptions for assigned methods.

```

Node::URDH_DIR_M_EQU() : Return Boolean ;
    Let MS ← URDH_DIR_M_A();
    Let NS ← equivalent_set(); -- return the equivalent nodes
    Iterate Over All Nodes Ni in NS;
        Let N_MS ← Ni.URDH_DIR_M_A();
        If N_MS  $\neq$  MS
            Then return False;
    End Iteration;
    Return True;
End Node::URDH_DIR_M_EQU;

Node::URDH_DIR_M_SUB() : Return Boolean ;
    Let MS ← URDH_DIR_M_A();
    Let NS ← subsumption_set(); -- return the subsumed nodes
    Iterate Over All Nodes Ni in NS;
        Let N_MS ← Ni.URDH_DIR_M_A();
        If N_MS  $\setminus$  MS  $\neq$  {}
            Then return False;

```


End Iteration;
 Return True;
 End Node::URDH_DIR_M.SUB;

- Checks equivalences/subsumptions for OTs accessed by assigned methods.

Node::URDH_DIR_OT_EQU() : Return Boolean ;
 Node::URDH_DIR_OT_SUB() : Return Boolean ;

- Checks equivalences/subsumptions for private data items accessed by assigned methods.

Node::URDH_DIR_PD_EQU() : Return Boolean ;
 Node::URDH_DIR_PD_SUB() : Return Boolean ;

Techniques for direct analyses of prohibited methods have also been developed, as well as indirect analyses for both assigned/prohibited methods.

References

- [1] R. Agrawal and N. Gehani, "ODE (Object Database and Environment): The Language and the Data Model", *Proc. of the 1989 ACM SIGMOD Intl. Conf. on Management of Data*, June 1989.
- [2] D. Bell and L. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", Technical Report MTIS AD-A023588, The MITRE Corporation, July 1975.
- [3] H. H. Bruggemann, "Rights in an Object-Oriented Environment", *Proc. of Fifth IFIP WG11.3 Working Conf. on Database Security*, Shepherdstown, W. Virginia, Nov. 1991.
- [4] S. Demurjian, H. Ellis, and M.-Y. Hu, "Software Reuse and Evolution in ADAM: A Joint Object-Oriented Programming Language and Database Design Tool", *Proc. of 1990 Sym. on Object-Oriented Programming Emphasizing Practical Applications*, Sept. 1990.
- [5] S. Demurjian, M.-Y. Hu, D. Kleinman, and A. Song, "ADAM/DDD - An Application-Specific Database Design Tool for Dynamic Distributed Decisionmaking", *Proc. of IEEE Systems, Man, and Cybernetics Conf.*, Oct. 1991.
- [6] S. Demurjian, T.C. Ting, and M.-Y. Hu, "An Analytical Framework for User-Role Based Security in an Object-Oriented Design Model", Technical Report CSE-TR-91-22, Department of Computer Science and Engineering, University of Connecticut, Aug. 1991; submitted to *VLDB Journal*.
- [7] S. Demurjian, M.-Y. Hu, and T.C. Ting, "Towards an Authorization Mechanism for User-Role Based Security in an Object-Oriented Design Model", Technical Report CSE-TR-92-2, Department of Computer Science and Engineering, University of Connecticut, Feb. 1992; submitted to 1993 Phoenix Conference on Computers and Communications.

- [8] D. Denning, et al., "Views for Multilevel Database Security", *Proc. of IEEE 1986 Sym. on Security and Privacy*, May 1986.
- [9] K. Dittrich, et al., "Discretionary Access Control in Structurally Object-Oriented Systems", in *Database Security, II : Status and Prospects*, C. Landwehr (ed.), North-Holland, 1989.
- [10] H. Ellis, S. Demurjian, F. Maryanski, G. Beshers, and J. Peckham, "Extending the Behavioral Capabilities of the Object-Oriented Paradigm with an Active Model of Propagation", *Proc. of the 18th Annual ACM Computer Science Conf.*, Feb. 1990.
- [11] H. Ellis and S. Demurjian, "ADAM: A Graphical, Object-Oriented Database Design Tool and Code Generator", *Proc. of the 19th Annual ACM Computer Science Conf.*, March 1991.
- [12] M.-Y. Hu, S. Demurjian, D. Kleinman, and A. Song, "ADAM/DDD - A Scenario Design Tool for Dynamic Distributed Decisionmaking", *Proc. of BRG 1991 Sym. on Command and Control Research*, June 1991.
- [13] T. Keefe, et al., "A Multilevel Security Model for Object-Oriented Systems", *Proc. of 11th Natl. Computer Security Conf.*, Oct. 1988.
- [14] W. Kim, "Object-Oriented Databases: Definition and Research Directions", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 3, Sept. 1990.
- [15] D. Kleinman and D. Serfaty, "Team Performance Assessment in Distributed Decision Making", *Proc. of Workshop on Simulation and Training Research Sym.*, April 1989.
- [16] C. Landwehr, et al., "A Security Model for Military Message Systems", *ACM Trans. on Computer Systems*, Vol. 2, No. 3, Sept. 1984.
- [17] B. Liskov, et al., "Abstraction Mechanisms in CLU", *Comm. of the ACM*, Vol. 20, No. 8, Aug. 1977; Also appearing in [33].
- [18] F. H. Lochovsky and C. C. Woo, "Role-Based Security in Data Base Management Systems", in *Database Security : Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.
- [19] T. Lunt and D. Hsieh, "The SeaView Secure Database System: A Progress Report", *Proc. of 1990 European Sym. on Research in Computer Security*, Oct. 1990.
- [20] F. Rabitti, et al., "A Model of Authorization for Object-Oriented Database Systems", *Proc. of the Intl. Conf. on Extending Database Technology*, March 1988.
- [21] F. Rabitti, et al., "A Model of Authorization for Next Generation Database Systems", *ACM Trans. on Database Systems*, Vol. 16, No. 1, March 1991.
- [22] P. Rougeau and Stearns, "The Sybase Secure Database Server: A Solution to the Multilevel Secure DBMS Problem", *Proc. of 10th Natl. Computer Security Conf.*, Oct. 1987.
- [23] J. Shilling and P. Sweeney, "Three Steps to Views: Extending the Object-Oriented Paradigm", *Proc. of 1989 OOPSLA Conf.*, Oct. 1989.
- [24] D. Spooner, "The Impact of Inheritance on Security in Object-Oriented Database Systems", in *Database Security, II: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1989.

- [25] P. Stachour and B. Thuraisingham, "Design of LDV: A Multilevel Secure Relational Database Management System", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 2, June 1990.
- [26] M. Thuraisingham, "Mandatory Security in Object-Oriented Database Systems", *Proc. of 1989 OOPSLA Conf.*, Oct. 1989.
- [27] T.C. Ting, "A User-Role Based Data Security Approach", in *Database Security : Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.
- [28] T.C. Ting, "Application Information Security Semantics: A Case of Mental Health Delivery", in *Database Security, III : Status and Prospects*, D. Spooner and C. Landwehr (eds.), North-Holland, 1990.
- [29] T.C. Ting, S. Demurjian, and M.-Y. Hu, "On Information Hiding for Supporting User-Role Based Database Security in the Object-Oriented Paradigm", *Proc. of Fifth IFIP WG11.3 Working Conf. on Database Security*, Shepherdstown, W. Virginia, Nov. 1991.
- [30] T.C. Ting, S. Demurjian, and M.-Y. Hu, "Requirements, Capabilities, and Functionalities of User-Role Based Security for an Object-Oriented Design Model", to appear in *Database Security, V : Status and Prospects*, C. Landwehr and S. Jajodia (eds.), North-Holland, 1992.
- [31] D. Tsichritzis and F. Lochovsky, *Data Models*, Prentice-Hall, 1982.
- [32] P. Wegner, "Concepts and Paradigms of Object-Oriented Programming", *OOPS Messenger*, Vol. 1, No. 1, Aug. 1990.
- [33] *Readings in Object-Oriented Database Systems*, S. Zdonik and D. Maier (eds.), Morgan Kaufmann, 1990.
- [34] S. Zdonik and D. Maier, "Fundamentals of Object-Oriented Databases", in [33].

Integrity and the Audit of Trusted Database Management Systems

Jarrellann Filsinger

Security Technical Center
Mail Stop Z231
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

(703) 883-5513

KEYWORDS: Database Management Systems, Audit, Integrity .

This paper is currently under review for public release.

July 14, 1992

INTEGRITY AND THE AUDIT OF TRUSTED DATABASE MANAGEMENT SYSTEMS

Jarrellann Filsinger

Security Technical Center
Mail Stop Z231
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

ABSTRACT

Research to date and current practice in trusted database management system (DBMS) audit have focused primarily on issues of confidentiality. In this paper, we address the need for auditing with respect to integrity. Although the TCSEC establishes policy for collecting data relevant to confidentiality, it gives little guidance on the audit for integrity. However, the protection of data integrity is an important aspect of DBMSs. In the paper we review the definition and purpose for audit and discuss the implications for integrity. Since the audit of integrity-related DBMS actions or events requires an examination of the context in which integrity is used, the paper uses the example of a medical information system to provide a rich context. This paper recommends that an explicit audit policy be established for a database system to ensure that both confidentiality and integrity concerns are thoroughly addressed. We describe the content of such a policy and give a framework to show how it would be applied to the medical example.

1.0 INTRODUCTION

In multilevel or trusted operating systems (OS), the broad emphasis of auditing has been on meeting the Accountability Control Objective as specified in the Trusted Computer System Evaluation Criteria (TCSEC) [TCSEC85]. Now, the same auditing requirements are being applied to trusted Database Management Systems (DBMS) as documented in [NCSC91d, Scha90, NCSC88c]. The Accountability Control Objective from the TCSEC states "Systems that are used to process or handle classified or other sensitive information must

assure individual accountability whenever either a mandatory or discretionary security policy is invoked" [TCSEC85]. The Accountability Control Objective is defined in such a way as to be applied to access control requirements for an OS and a trusted DBMS; however a trusted DBMS must further consider integrity requirements. A trusted DBMS must not only adhere to the Mandatory Access Control (MAC) and Discretionary Access Control (DAC) policies, but also to an integrity policy. Subsequently, audit for a trusted DBMS must be comprehensive in order to meet all three policy domains.

For the purpose of this paper, integrity is concerned with two concepts: data integrity as it is applicable to the objects in a DBMS, and system integrity, which relates to the DBMS in its environment. Data integrity is related to the type of audit that is automated and internal to the DBMS, whereas system integrity is more closely associated with the type of audit that is conducted external to the DBMS and involves other automated and unautomated components.

To illustrate the distinction between integrity audit and security audit, an example is taken from the health care industry. Since medical Automated Information Systems (AISs) containing DBMSs are complicated applications, it is not sufficient to characterize accountability as "the audit of security-relevant events." Instead, a statement of the intended audit goals that addresses the concerns of both security-related audit and integrity-related audit is needed to capture robust DBMS protection requirements. An audit policy can express these audit goals, objectives, and requirements. An audit policy plays a role similar to that of a security policy in a trusted system; together, they define the philosophy of protection for a particular implementation.

The Decentralized Hospital Computer Program (DHCP) delivers health care services to eligible veterans as one of the primary missions of the Department of Veterans Affairs (VA). DHCP does not have a multilevel security requirement in the Department of Defence (DOD) context, however, an argument could be made for hierarchical classification of data due to need-to-know restrictions. The DHCP is a rich source of complex data relationships ideal for the study of data and system integrity.

The remainder of this paper is organized as follows. Section 2 gives the definition and purpose for audit in a trusted DBMS. The purpose for audit is related to both integrity and security. Section 3 presents a discussion of accountability that incorporates both access control security objectives and integrity objectives. The general nature of an audit policy is presented in section 4. The DHCP system described in section 5 provides background information for the DHCP audit policy framework. The guidance for an DHCP audit policy illustrates how integrity audit objectives can be addressed in a medical information system. Section 6 concludes by reviewing the key points identified in this paper.

2.0 AUDIT DEFINITION

A broad definition of audit is that it is the collection and review of a documented history of the use of a system to verify that the system is intact and working effectively. The auditing activity, viewed in its entirety, encompasses the notion of both an *external* and *internal* audit.

An *external audit* consists of gathering and analyzing documentary evidence about the system by methods which are external to the system. External audit includes an extensive review of a system that covers an assessment of system security policy, data integrity controls, system development procedures, and back-up and recovery procedures. External audit is an in-depth audit of a particular system for accuracy. An auditor inputs special

transactions and monitors their progress as they flow through the system. Verifying the correctness of the transaction flow provides evidence to certify the accuracy and accountability of the system. The external audit process and the collection of documentary audit evidence provide assurance that the system under inspection is functioning properly.

An *internal audit* is distinguished from an external audit by the fact that, whereas an external audit is only performed periodically and by means external to a system, an internal audit is designed into the system and runs continuously. Internal audit consists of automated collection and analysis of documentary evidence of the system's use. An internal audit subsystem must include mechanisms for continuously collecting and recording audit information in chronological order and for periodically reviewing and analyzing the collected information. The internal audit collection mechanism records information about system activities that have been judged significant enough to audit, which are referred to as *audit events*. An audit event may originate in the operating environment (e.g., a network connection to a DBMS), in the operating system (e.g., as a file open), or in the TDBMS (e.g., as a database query).

Information collected about the occurrence of audit events is recorded in an audit trail. Since it serves as the only internal accountability mechanism, the audit trail must be tamperproof and must be afforded protection equivalent to that provided for the documentary evidence used in external auditing.

The health care industry is well acquainted with the auditing process. The term *medical audit* has a distinct meaning:

Medical audit is a retrospective review by medical staff members of selected hospital medical records, performed for the purpose of evaluating the quality and the quantity of medical care in relation to accepted standards [HAT89].

For a health care environment in which medical records are not computerized, medical audit is associated with what has been defined here as external audit. Medical audit is a catalyst for the move to computerization of medical records [Nixon90]. As the health care industry comes to depend on AISs, then medical audit will most likely include requirements for internally auditing computerized medical records. If medical audit becomes the driver for automated medical records, then audit generated from a medical DBMS must contain information that is not strictly related to security or privacy of the DBMS, but for integrity as well to support the objective of evaluating medical care.

2.1 PURPOSE OF AUDIT

The purpose of audit is to allow an authorized agent to review a history of events taking place on the system in support of the accountability control objective [TCSEC85]. Trusted operating systems built to the requirements of the TCSEC are concerned with the disclosure of information and support a security policy whose emphasis is confidentiality. A trusted DBMS has an additional concern, that of integrity, and it must enforce integrity properties in addition to the nondisclosure policy.

The audit subsystem is generally structured in two parts: the audit mechanism itself, which does the actual collection and posting of an audit event to the audit trail; and a reduction facility or tools to aid in the analysis and review of audit trail data. Both parts together are needed to meet the audit objectives.

The *Guide to Understanding AUDIT in Trusted Systems* [NCSC88c], hereafter referred to as the *Guide*, incorporates the TCSEC accountability control objective and provides an interpretation of the TCSEC audit requirements. It amplifies the purpose for trusted computer system audit given in the TCSEC, providing five specific objectives satisfied by audit. However, like the TCSEC, the *Guide* discusses audit in terms of its support for the confidentiality control policy. Each of the five specific audit objectives stated in the *Guide* is restated below, with extensions added to include both confidentiality and integrity concerns.

1. **Allow for reviewing patterns of access** - In order to allow a complete audit analysis for a DBMS, the appropriate audit information must be retained in the audit trail. A Medical information system (MIS) has the need for confidentiality of medical records and for assurances of the integrity of those records. To address both confidentiality and integrity policies, database read operations are audited to support a confidentiality policy, whereas write or modification operations are audited to support both the confidentiality and integrity policies.
2. **Allow for discovery of attempts to bypass system controls** - Given that adequate audit information has been collected, the ability to detect policy violations also depends on the capabilities of the tools used to review and analyze the audit information. In order for a DBMS to meet this objective, the tools must support the capability to discover attempts to violate both confidentiality and integrity policies.

In an MIS, control over access to patient medical records is provided by the attending physician who is sworn to uphold the Hippocratic Oath¹. An audit record would provide the history of accesses and updates to patient medical records necessary for the protection of confidentiality and integrity. Intrusion detection tools are needed in MIS to serve as monitors for the protection of patient privacy and preserve integrity through identification of unauthorized modifications.

3. **Allow for discovery of use of privilege** - Audit must record the use of privilege to support the TCSEC accountability control objective. Execution of a privileged command permits a subject to violate a system's security or integrity policy; therefore, these become critical functions to audit. The privileged command set is primarily used by those administering the system who are permitted to violate a system's security or integrity policy. The use of a DBMS's privileged commands must be accounted for in the audit trail. One of the utmost concerns in health care is limiting professional liability exposure, as hospitals face increasing liability for the actions of its employees and physicians, physical records such as the audit trail provide proof of the use and abuse of privilege.
4. **Act as a deterrent** - As in an OS, the mere knowledge of the existence of an audit capability in the DBMS is an important element in deterring malicious behavior. The deterrence objective is met by the provision of auditing for both a confidentiality and an integrity control policy. With many functional departments

¹ "Whatsoever things I see or hear concerning the life of men, in my attendance on the sick or even apart and herefrom, which ought not to be noised abroad, I will keep silence thereon, counting such things as to be sacred secrets." [LICH86].

of a health organization becoming integrated within the boundary of one AIS, it is essential that the audit of DBMS medical records be a well known fact in the user population.

5. **Provide additional assurance** - The assurance objective of the audit mechanism applies uniformly to both trusted OSs and trusted DBMSs. It is important in providing the ability to reliably document bona fide policy violations. The audit mechanism is relied upon to record all significant system activity and nothing else. The audit mechanism should be trusted to record only the significant system activity prescribed by a system administrator or designee. Audit provides proof of accuracy among medical records. Audit is an additional assurance of quality for an MIS.

The audit of integrity encompasses both data integrity and system integrity. One of the broad goals of integrity is to maintain internal and external consistency among the data in the database and the reality they represent externally. The Clark and Wilson Integrity Model [Clark87] introduces the concept of a Transformation Procedure (TP), which is an integrity mechanism that is built into the system and meant to preserve internal and external controls. The audit trail is a double check on the controls provided by the TPs. Through the collection and review of audit data, an internal and external view of the data can be maintained. Internal and external consistency are system integrity controls that are attained by the DBMS audit indirectly through the analysis of audit data, as stated in objective 2. That is, internal and external consistency are byproducts of the audit trail analysis, rather than the audit activity itself.

If the DBMS audit mechanism provides a comprehensive record of system actions, then analysis of audit data can detect violations of internal consistency controls. Data integrity constraints add to the richness of the DBMS semantics by two methods: one is structural and the other is content-based. The structural integrity constraints concern only equalities among values in the database. The most prevalent of this type of integrity constraint is called a *unique key constraint*. The unique key constraint defines a set of fields or attributes that form a unique key for the record or relation. When a column within a table is specified as the unique key, then, typically constraints on those fields values are also stated.

The second content -based method a TDBMS uses to enforce integrity concerns controlling the actual values stored in the database. There are three types of constraints over actual database values: *domain*, *entity*, and *referential*. Most DBMS languages support a restricted set of domain types: fixed length character, fixed point number, integers, etc. In addition, domain integrity constraints allow for the specification of a valid set of data values. They are used to allow more precise control over the values that will be allowed on an UPDATE or an INSERT. For example, a patient cannot have an age greater than 110 in the age field of the patient record. *Entity* integrity states "that no primary key attribute of a base relation is allowed to have null values. (Null values represent "information unknown.") The entity integrity rule is justified because, by definition, entities in the real world are distinguishable (i.e., they have a unique identification of some kind) [Date90]. *Referential integrity* affects the actual data values in the database. It guarantees that values appearing in one table for a given set of attributes, also appear in another table. For instance, for every doctor assigned a patient in the patient's file; the doctor's name and associated information must exist in the doctor's file. While these referential integrity "linkages" help to maintain a consistent view of the data, in order to enforce this property, the DBMS must modify tables without explicit instruction from the end user. The DBMS must be trusted to enforce this property correctly. As is to be expected, the audit of the actions taken with respect to referential integrity should reflect all data modifications. Since

even a small data modification can result in several additional modifications required to enforce the integrity constraints. The result of one command may produce many audit records. Today, many commercial DBMSs minimally implement domain integrity constraints and very few enforce referential integrity.

Separation of duty is an external consistency control useful for maintaining system integrity. External consistency controls are implemented most commonly by separation of duty methods: where an action is broken into subparts and each subpart must then be executed by a different person. The audit mechanism may be used to detect deviations of separation of duty controls; specifically, the *order* in which each subpart or subtransaction is executed and who executed it can be detected by audit trail analysis. Karger [Karger88] suggests the role of the audit trail is not just for security audit, but for the implementation of separation of duty as described by the Clark and Wilson (C&W) Model [Clark87]. The separation of duty (C&W rule C3) can only be enforced by using information about past actions. Historical information governing past actions, that is the action and the subject of each action, is found in the audit trail.

Another goal of integrity is to prevent authorized users from making unauthorized or improper modifications to data. Mechanisms, like the audit trail, are usually provided to minimize the risk of this type of integrity violation. Objectives 2 and 3, the discovery of attempts to bypass system controls and the audit of the use of privilege mechanisms, are applicable to this integrity goal. Medical staff, such as physicians, are given specific clinical privileges according to the hospital policies, and must have these privileges renewed at certain intervals [AAP90]. The clinical privilege granted by the hospital administration is mapped to the access control policies enforced in an AIS. The audit trail is one mechanism that can be used to provide assistance in monitoring the actions and privileges of the medical staff to ensure that the activities result in quality care for the patients.

Additional system integrity objectives are met through objectives 4 and 5. The deterrence and assurance audit objectives fulfil system integrity goals rather than data integrity goals, because these audit objectives relate to the DBMS environment. The competitive environment of today's health care industry, coupled with the rising malpractice verdicts, defense costs, and insurance premiums, has forced hospital administrations and physicians to respond by placing increased emphasis on systems to monitor, promote, and guarantee quality service. An MIS is used to promote a higher quality of health service. The deterrence and assurance goals of the audit of MISs serve to aid system integrity as a means in attaining a higher standard of health care delivery.

Audit in a medical information system may have many purposes that can be met by the five objectives described above. A hospital information system, for instance, collects specific information for a medical audit (as previously defined). That is, the audit information collected becomes the means by which the quality of care is assessed, performance (of health care professionals) indicators are derived, liability is reduced, and the interrelations among fundamental hospital components are examined. It is clear that audit can be used to meet a broad range of requirements, when integrity audit events are included in the audit trail.

3.0 ACCOUNTABILITY

The audit trail is the only physical record of DBMS accountability. System integrity must figure prominently into the accountability objective since integrity concerns are, like accountability, related to the external environment. Both integrity from a data sense and

from a systems sense is needed for accountability. The technical report, *Integrity in Automated Information Systems*, addresses one aspect of system integrity in the definition it provides on accountability:

... accountability. .. requirements [that] are derived to uniquely identify and authenticate the individual, to authorize his actions within the system, to establish a historical track or account of these actions and their effects, and to monitor or audit this historical account for **deviations from the specified code of action** [NCSC91b].

This definition of accountability incorporates the notion of an external standard code or policy to which an individual is expected to conform. Many organizations contain a code of good conduct. The medical profession has very stringent codes of conduct. "Medical staff credentialing, including the delineation of clinical privileges for each staff member, represents a cornerstone in the hospital quality assurance program [AAP90]." The entire medical credentialing process must be clearly stated in the hospital staff medical bylaws. Given such a firm foundation in an external code of conduct, the task at hand is to represent that code in a policy the DBMS can enforce. An audit policy can serve, in part, as a representation of the external hospital policy. If specific actions, such as updating the medical record each time the patient receives care, relate to an external code of conduct that can be audited, such deviations from the code can be detected. The audit policy can address both data integrity by delineating the level of accountability, and system integrity by incorporating the external code of conduct.

In summary, since accountability is a product that is derived from the physical record of end-user actions it should incorporate both data integrity and system integrity attributes. To ensure that the audit is complete and comprehensive, it is necessary to make the objectives for audit a formal part of a system's protection strategy. This requires documenting these objectives in the form of an audit policy.

4.0 AUDIT POLICY DEFINITION

A health service is like an organism that needs a central nervous system to cope with the complexities of modern information handling. Without a policy for information handling, data systems become patchy and idiosyncratic — an opinion that may not be unfamiliar to those trying to draw conclusions from existing data [Seddon90].

An audit policy is a statement of high-level rules, goals, and practices that describes how the organization manages and protects its audit data. The audit policy encompasses technical, administrative, and procedural aspects of DBMS audit. The technical aspects of the audit policy will define *what* objects are important for audit (for security and integrity) and *when* actions against these objects should be audited. Not every object or class of objects will have to be audited at all times. The set of auditable objects may vary depending on the level of risk and the perceived threats. The policy should be sufficiently detailed so it can be tailored to the technical capabilities of the target DBMS. A technical mechanism important for the enforcement of an audit policy within a DBMS is the ability to define audit options associated with data objects. The audit policy can help to delineate which audit options are to be used: part, some, or all of the time. Once it has been determined what objects are to be controlled and which actions are security-relevant and which are integrity-relevant, then the DBMS audit options can be set to optimize the audit resource.

Selectable audit events are provided by the DBMS as a way to balance the trade-off of too much unintelligible audit information against too little or none.

The administrative part of an audit policy addresses organizational values. This section of an audit policy may be used to define the risks the audit attempts to minimize. An organization-wide audit policy may state the overall purpose and intent of audit, but each functional unit will be interested in accountability over its own data objects. For example, a hospital, as an organizational entity has an interest in maintaining one level of accountability, while organizational subparts, such as pediatrics or finance, may have separate or mutual exclusive accountability goals. An audit policy attempts to provide some adhesion to the different perspectives of audit. The administrative criteria also must establish the scope and coverage of the audit with the associated costs of the audit. The costs of audit include the audit review time, audit collection, and audit storage media, in addition to the DBMS performance costs.

The audit policy contains details on the procedural aspects of audit data for its use once it has been collected. The importance of protecting the audit data is paramount. The audit policy can direct who is to view the data, how often it must be reviewed, key data points to be gathered, and, most important, what to do when anomalous events are detected. Several procedural and operational system management requirements can also be delineated in the audit policy, such as, the handling of audit data and maintenance of records pertaining to the storage of the data. The movement of online audit trails to offline storage is another facet of the policy. The audit policy should define an archive schedule for labeling and storing audit trail data offline. Audit trail maintenance information is needed for the retrieval and analysis of audit information stored offline.

5.0 HEALTH CARE SERVICE APPLICATION

This section applies the concept of an audit policy to an application. The DHCP was chosen because the DHCP is one of the few MISs that supports multiple medical centers and incorporates clinical and administrative information. The following paragraphs describe the DHCP functionality, and then the framework for an audit policy is presented. Although, security plays an important role in audit, the purpose of this paper is to define the integrity view of an audit policy for a health care organization.

5.1 THE DHCP

The delivery of health care services to eligible veterans is one of the primary missions of the Veterans Administration (VA). AISs are indispensable resources to the delivery of quality health care to the vast numbers of veterans. The Veterans Health Administration (VHA) operates the largest centrally directed health care system in the United States (US) through 172 VA Medical Centers (VAMCs), 229 outpatient facilities, 122 nursing homes, and 27 domiciliaries. During Fiscal year 1990, these facilities attended 1.3 million inpatient and 22.6 outpatient visits. Because of the need to serve U.S. veterans, the DHCP is in a unique position for creating standardization in the field of medical informatics [Mun86]. Due to its success, the DHCP is being used as a model hospital information system. The DHCP system is also being adapted for use by the Indian Health Service, and the DOD has implemented the system on selected military bases. The Helsinki University Central Hospital, in Finland, has also implemented the DHCP.

The focus of the DHCP has been to develop a set of core modules that was easily integratable into a complete hospital information system. The key architectural features of the DHCP are listed below:

1. Single standard language - MUMPS (Massachusetts General Hospital Utility Multi-Programming System). MUMPS is an American National Standards Institute (ANSI) language.
2. Common database - The set of common databases is controlled by an active data dictionary. These data dictionaries are designed to allow for both centralized and decentralized control of the data. This common database environment is provided by FileMan, which is a collection of MUMPS routines written by the VA, but is available now as public domain software. FileMan is fundamentally a hierarchical data management system; however, it includes a pointer data type and a trigger mechanism common with relational DBMSs [Dav87a].
3. Host and communication services - Digital Equipment Corporation (DEC) equipment is networked through an X.25 packet-switching system.

There are three major areas of the DHCP: the system/database kernel, the clinical, and hospital administration applications. Each area of the integrated hospital management system is described below. The DHCP system/database management function is structured as a set of kernel programs that provides an interface between MUMPS applications and the OS. The major portions of the system/database management kernel are:

1. The file manager (FileMan) which includes the DBMS and supporting utilities such as a report writer, data dictionary, and data editor.
2. Electronic mail (E-mail) provides user-to-user teleconferencing, networking, and software distribution.
3. The task manager (similar to the Unix cron program) initiates time-sensitive background tasks.
4. The menu manager controls the user access to functional elements of the DHCP and provides online help.
5. The security monitor controls user identification and authentication, locks out devices, tracks usage, and controls which menus a user can access.
6. The OS configuration data are managed by a table designed for each OS on which the DHCP is implemented.

The clinical application area contains the set of core medical databases. The DHCP contains many other clinical modules; however, not all of these other modules are distributed nation-wide. Only the core databases are resident at every VA medical center. The core medical modules are listed below:

1. Admissions, Discharge and Transfer (ADT). This application collects and tracks inpatient activity, collects patient demographic information, bed assignment, patient release (discharge), and waiting lists. The ADT also prepares patient classification information that is used for assessing the cost of the services provided and for developing management reports for utilization review.

2. **Outpatient Scheduling.** This application establishes clinical appointments. The appointments are scheduled to minimize travel time and expense of patients by scheduling multiple appointments on the same day.
3. **Outpatient Pharmacy.** This application allows pharmacists or technicians to manage outpatient prescriptions. It has a suspense system to hold prescriptions until 10 days before they are due, and has 30 user-controlled site parameters to allow local pharmacist control over the system when local policy does not conflict with national policy. The package does automatic checks for duplicate drugs and displays allergies and other clinical-relevant information. Reports are established for drug utilization review, physician prescribing practices, and cost.
4. **Laboratory.** The clinical lab supports the collection and storage of lab test information. Retrieval of this information is restricted to authorized lab, ward and clinical personnel. The lab tests may be entered manually or through automated interfaces with lab equipment. All information is verified before being entered permanently in the patients medical record. The laboratory module can accept orders for tests from the lab or the ward. It also prints worklists that help the staff track test specimens and the tests to be performed with the specimens. Additionally, the system can display comparative analysis of data, flag abnormal, and critical values, and prevent the release of information if controls or instrument calibrations are out of acceptable ranges. Reports generated include test descriptions and requirements, lab workload, tests in progress, and data related to a single patient's lab tests for a specified time period.
5. **Inpatient Pharmacy.** The inpatient pharmacy application is composed of three functions: unit dose, ward stock and IntraVenous (IV) drugs. The unit dose supports inpatient drug distribution while the patient is in the hospital. Orders are entered, reviewed, or canceled by appropriate medical personnel. The unit dose reports patient demographics, all active orders, and administration schedules. Ward stock or automatic replacement of supplies handles the distribution and drugs inventory for the hospital. The IV function is a dispensing package that provides the pharmacy users with drug labels, manufacturing data, and basic counting or reporting of IV orders.

The management modules contain the hospital administration, decision support and financial applications. A selected few management applications are described below.

1. **Accounts Receivable.** This application is a bookkeeping function that tracks the monies that are due to the VA.
2. **Decision Support System.** This application consists of a set of data extract routines that capture data from other clinical and management DHCP applications. The data are used for modeling, variance analysis, and ad hoc reporting.
3. **General Inventory.** This application is designed to provide the basic inventory tasks such as ordering, receiving, and distributing hospital supplies. This application is fully integrated with accounting and procurement applications.
4. **Interim Management Support.** This application focuses on staffing, costs, materials, and space resources. ~~The management functions are~~ targeted for the VA medical center director, the associate director, and the chief of staff.

The DHCP has implemented a global data model that is centrally controlled, but implemented and modified according to local medical and legal requirements. Information flows horizontally from to regional local medical centers, and in some cases, to national data processing centers. Local medical centers and regional data processing centers also exchange data; thus, information flows vertically as well.

A two-tiered data architecture within the DHCP is used for some applications and other applications involve a three-tiered data architecture. The database applications are developed and maintained centrally at a national level. Applications and data are distributed to each VA medical center for use and modification on a local level. Some application areas are also required to maintain consistency with national-level databases. The Tumor Registry application allows personnel to abstract cancer cases from the national-level for use in treatment and provides utility options for file maintenance, and data consistency with the National Registry for online changes. A three-tiered data hierarchy involves the Regional level data processing centers within the VHA. The Quality Improvement application is designed to enhance the VA's accountability with respect to quality health care and enhanced management of resources. A quality checklist is submitted from a local medical center and forwarded to the national database, where it is compiled and analyzed. Results are then reported to the VA Central Office which ultimately provides feedback to the local level. Information within this application is extracted and reviewed from databases at three levels. Information is also exchanged between VA medical centers. An Automated Medical Information Exchange (AMIE) application facilitates information exchange among several VA components: the Veterans Benefits Administration (VBA) at the national level, the regional offices and each local VA medical center. Information is exchanged for benefit eligibility, status of pending examinations and tests, confirmation of payments, and benefits adjustment.

The basic security and integrity features provided in the DHCP are built and maintained within the kernel system software. Security management includes access controls of users to resources (e.g., programs, menu options, files, fields with files, and devices). These access controls can be specified by user, device, time of day, and day of week. Before an application is distributed to a local medical center, the access controls for the application are specified by the local medical center, but implemented in the kernel (FileMan) at the central control distribution facility.

The FileMan software controls access based on user identity. A user identifier is associated with a file, and within the file, to a field. The DBMS functions are mapped to the controls implemented by FileMan in table 1. These access controls are used for the file level; access at the field level is limited to Read, Write, and Delete. Access to each DBMS file or field is controlled by a "permission key" established when the file is created. A user must have the "key" to the data dictionary in order to set field level granularity access control.

DBMS Functions	FileMan Options Controlled
Read	Print, Search, Inquire, Statistics, and List File Attributes
Write	Enter/Edit
Delete	Enter/Edit and Transfer File Entries
Data Dictionary	Modify File Attributes and Utility Options

Table 1. Granularity of DBMS Access Control

The effect of creating file and field level granularity is that a user is presented with a specific *view* of the application data. Each clinical and hospital management application has the ability to be configured for a user's privileges. The data and the application are tightly coupled and control is provided using a view. An example of the use of the access control mechanisms is the Mental Health application menu which contains several functions, including clinical record, patient-administered instruments, vocation rehabilitation, general management, inpatient functions, and Mental Health Service (MHS) manager functions. The MHS manager functions allow the designated manager (as set by the DHCP kernel) to administer permissions. The MHS manger menu contains functions for configuring the inpatient features, the local site management parameters, individual permissions for field-level control of the clinical record, and access to E-mail. This example of view-based DAC heightens the need for a comprehensive audit criteria.

The integrity features within the DHCP include domain, entity, and referential constraints. Data fields are configurable to allow a preset list of values. For example, the field CITY can be programmed to accept only a list of local cities in an area. The VERIFY FIELDS is a utility that helps to identify entity and referential integrity constraints. The utility locates values in the file that are inconsistent with the definition of the element in the data dictionary [Dav90b]. To identify referential integrity constraints, the utility seeks database pointers that are not pointing to a valid data element. These dangling references are not prevented from occurring in FileMan; they are only identified after the fact, when the utility is executed.

The audit of integrity features performed by FileMan will be an inadequate subset of what is required to support a DHCP comprehensive audit trail that supports both confidentiality and integrity. Until referential and entity integrity constraints are incorporated into the DBMS product, much of the integrity-related audit information is found in the DHCP applications. The actual audited actions implemented in the DHCP are unavailable (at this time); therefore the following assumptions concerning audit have been made:

- a. Audit data are collected and used at the local level, and some relevant audit information may be transferred to the regional or national levels.
- b. Audit records contain an identifier, time, location, object name, the action taken with the object, and status of the event (success/fail).

- c. Audit events are created as a result of information transfer out of or into a local VA Medical Center.
- d. Audit events are created as a result of access controls placed on files and fields within files (DAC-based views).
- e. Manipulation of the data dictionary is audited.

The semantics for integrity-related audit are found in the DHCP applications. Nevertheless, many individual operations on data are performed transparently to the user and should be audited.

The DHCP is an evolutionary system addressing many health care administration technological issues. Its extensive clinical and management features are still being developed. Application areas, such as medical knowledge bases and research and development with universities and teaching hospitals, have yet to be exploited. Many technology areas, such as improved clinical displays and workstations along with software development tools, are areas identified for future growth. With a continuing interest in the field of medical audit brought on by the increasing need for medical record review, audit is likely to be considered an area for future growth as well. The framework for the development of an audit policy is just one step needed to address the complex policies in a large, integrated, hospital medical information system such as the DHCP. A fully developed audit policy requires very specific knowledge and research into the DHCP, but it will provide a good foundation for maximizing the audit capability.

5.2 FRAMEWORK FOR A DHCP AUDIT POLICY SUPPORTING INTEGRITY

The framework and guidance for an audit policy is described below. The audit policy initially requires the statement of the DHCP audit goal and purpose for audit. To address the goal and purposes, the framework for the audit policy is divided into three areas: technical criteria, administrative criteria, and procedural criteria. Accountability, confidentiality, system and data integrity are discussed within the technical portion.

Audit Goal

The DHCP audit policy is based on a statement of the goal or goal to be achieved through the audit of the DBMS. The goal of audit is derived from the more encompassing DHCP goal of attaining quality health care while minimizing the costs.

Audit Purpose

Purpose of the DBMS audit is to support individual and legal accountability, integrity and confidentiality among all the local VA medical centers, and the regional and national data processing centers.

To obtain the goal and purposes set forth, the audit policy must be developed together with risk management and quality-control plans. Audit is a natural extension of risk management. Risk management programs are designed to identify and reduce actual or potential risks to patient safety, so that patient care is improved and negligence claims are restricted. Higher risk areas such as, obstetrics, trauma centers, burn units, psychiatry,

and intensive care, are prone to be clinical areas that require close attention when the decision is made to identify auditable objects. In the DHCP, each application area should be assessed to determine the appropriate audit. In consideration of the many VA medical centers, additional focus must be given to the requirements for each facility. The purpose of an audit in one facility may not be critical to another. Each facility should have a common audit goal, but the risks may vary. The strength of audit lies in the local implementation, but the benefit is shared. The purpose of audit is a variable dependent on application, site, and the associated risks.

Technical Criteria

The technical criteria identifies potential auditable events that pertain to the DBMS and its application. As stated above the, criteria are dependent on the associated risks, the site and the particular application area.

Accountability

The audit for accountability is based on the relationship of the DHCP end user and that of the information required by the end user to perform a job. In TCSEC terminology, this is related to subjects and objects. The accountability-related audit data includes the following:

1. Identification and authentication (I&A). I&A parameters uniquely associate the user with a role and data. The environment or profile of each database user includes attributes and default access settings. The creation and maintenance of the user environment must be part of the audit paradigm.
2. Functional roles. The DHCP roles are defined per application/database area. Within each application, one or more roles may be defined. The number of roles within an application/database varies depending on the requirements of the application. Some applications process sensitive patient data and therefore, require a greater level of accountability. It is assumed that every user of the DHCP interfaces with the system in a role capacity.
3. Information. The information (objects) the user seeks is processed at many levels of abstraction, e.g., application/database, data dictionary, table, view, form, or a report. The granularity of the audit is a function of the control required on specific information. The accountability and control may vary according to the application, the role defined for the information, and dynamic circumstances such as the context of a particular operation, for example date/time. Named objects on which privileged subjects may perform operations must be identified.

The implementation of roles in a DBMS enhances accountability and eases the audit burden. The implementation of functional roles is also used to support separation of duty controls. When a user's job functions are carefully defined and controlled under the umbrella of a role, then audit for the purpose of revealing "patterns of access" is more predictable; therefore, variations of access can be identified. The degree to which commands and roles overlap has a strong correspondence to how narrow in scope a role is defined. A very fine construction of functional roles (more users having the same privilege) will create a wider accountability overlap; hence, more privileged data will be held by more users in the performance of some tasks. While this increases accountability from a functional aspect, it also has the effect of making the audit of privileged operations less unique. For example, when several end users are identified to assume a role that

includes the printing of a patient's medical record, then the output action is not as unique an action as it could be if only one person were assigned to the role.

The audit for accountability focuses on the subject of the action rather than the object of the action. The remaining three areas of confidentiality and integrity focus on audit of the data object.

Confidentiality

With computer-based medical records, medical personnel can be restricted to obtain only parts of the medical record that is germane to their responsibilities. Very strong confidentiality controls obtained through MAC are not available on the DHCP. The confidentiality that is available is auditable through the VA kernel utilities and within the application modules themselves. The audit collected for confidentiality should include the following information:

1. MAC (when available). Access control based on labeled data is not currently implemented in DHCP.
2. DAC implemented by the DBMS and application.
3. Privilege. The auditing of privileged operations provides assurance that the controls implemented through privileged actions are being used properly. A privileged action is one that is authorized to violate a DAC, integrity, or other system enforced-policy.

Sensitive data within the DHCP must be identified. Parts of the DHCP databases that contain this data or data that can be derived from sensitive data should be audited commensurate with the level of risk. With medical information systems, the risk of disclosure is based on the identity of the patient. Confidentiality can be protected by the minimization of the amount of sensitive information revealed in case the patient is identified. Therefore, the identity of the patient should be masked if written in the audit trail.

System Integrity

System integrity audit is related to the external consistency requirements for the DHCP in its environment. System integrity-related audit includes the following information:

1. Inter- and intra-DHCP organization data.
2. Inter- and intra-DHCP clinical and administrative data.
3. Data required for a medical audit.
4. Data supporting separation of duty.

The audit trail from the DBMS should support the requirements for external quality control, such as medical audit. Specific information for medical audit includes audit events which could be used to derive the quality of clinical services and treatments that actually produced the most effective outcomes. The information ratio of services and treatment to effective outcomes is important for maintaining and setting standards of care.

Data Integrity

The audit of integrity constraints enforced within the DBMS and within the application modules will provide the means to assess the quality of data integrity. Audit for integrity purposes, must be an outcome of the application, as it is not sufficient from the DBMS itself. The audit for data integrity purposes includes:

1. Domain, entity, and referential integrity constraints. This audit is aimed at the identification of integrity and consistency problems on a semantic level.
2. Triggers or similar mechanisms (e.g., FileMan logical file pointers) that are used to maintain data consistency at different levels of abstraction in the DBMS. Triggers may be employed for maintaining consistency between databases existing between sites and between database tables at the same site.
3. Audit of externally defined medical terminology. The MUMPS language exploits the use of medically coded phrases and lists of standard terms. The possible relationship between those phrases and lists within the program application creates a consistent interpretation of medical terminology within the DHCP. Clinical applications external to the DHCP may not conform to the precise definitions and would be candidate information for integrity-related audit.

Administrative Criteria

The administrative criteria within the audit policy addresses the policy decisions made concerning the scope and coverage of the audit. Three criteria are noted:

1. All services that affect the care of patients will be audited. Preplanning precisely which characteristics are to be audited is associated with risk management planning. Audit system designers should avoid the tendency "to create their own special purpose [audit system] designed only to satisfy the initial requirements" [Bony81].
2. The audit trail review process will be conducted performing an analytical review of the data for both confidentiality and integrity purposes. The results of the reviews affect the quality of patient care and its delivery.
3. Assistance to quality assurance programs. The product of the audit trail may contribute to quality assurance committees on pharmacy, nursing service and administration, medical record review, infection control, blood utilization review, and antibiotic review.

Pertinent to this policy decision concerning the scope of audit is the determination of resources available to manage audit data. Considerations for this decision concern the time, cost, and storage resources that are required for administration of audit data. The overall cost in terms of performance and storage requirements must be commensurate with the benefit from the audit. A performance versus audit tradeoff study conducted periodically will maintain an acceptable level of audit. Administrative decisions with respect to audit trail storage will be reflected in how easily audit trails are able to be retrieved and analyzed retrospectively. The storage records pertaining to archived audit trails must conform to strict quality-control standards. Audit trails may be used for legal accountability and therefore, must be maintained systematically without error.

In addition to specific management of resources, advanced technology in audit trail analysis and intrusion detection tools can help maximize the benefits of retaining and using audit trails when needed for reducing audit and detecting suspicious behavior. When the audit includes integrity-related events, audit trail analysis becomes even more important. There are more events of interest in the audit trail and the audit events are not collected solely for detecting system intruders. Different types of audit trail review are needed. The review of the audit trail will require sophisticated tools to take full advantage of the data collected.

The product of the audit trail review may extend into quality-assessment activities. The identification and assessment of the extent of observed problems in the areas of confidentiality, system, and data integrity will provide feedback into the quality-assurance planning activity in the DHCP organizational hierarchy.

Procedural Criteria

The procedural criteria for audit data are related to the actual guidelines and steps needed to implement the policy. Standards and operations guides are used for reviewing, maintaining, creating, and protecting archived audit data. The standards should address the frequency and type of audit trail review that will be performed. The frequency and type of review may be commensurate with the associated risks. Maintaining and creating audit trail data both on-line and offline are conducted according to prescribed standards. The protection of audit trail data encompasses online and offline storage. The protection of audit trail data should be addressed in contingency planning activities. The audit procedural criteria reflect the standards and operations between VA medical centers, and regional and national sites.

5.3 THE DHCP AUDIT POLICY SUMMARY

The audit policy attempts to provide a framework in which to study the specific objectives for audit within the DHCP. An audit policy will ensure the DHCP and VA administrators that *what* is being audited is done for an intended purpose. In order to create a fully effective audit policy, it is necessary that it be strongly coupled with the risks and vulnerabilities of the system. Audit can serve many purposes including confidentiality, and both system and data integrity. Some of the areas identified for audit are overlapping. Because "absolute" integrity is impossible to achieve, audit from several perspectives is provided to minimize risks. The policy, while, not complete at this point, requires more in-depth study of the information flow from within the application modules and between organizational components that support the DHCP. A long term study of the DHCP can yield audit principles applicable to similar medical information systems.

6.0 CONCLUSION

A DBMS audit mechanism is used to support both data integrity and system integrity. Data integrity serves, as an objective, to control the modification of data within the DBMS. System integrity is not so much related to the mechanism as it is to internal and external consistency requirements -- the view of the data with respect to real-world actualities. Integrity is tied to an application [NCSC91], and to analyze integrity, it is convenient to examine the integrity constraints of an existing system, such as the DHCP. An audit policy is required for a DBMS due to the complexity of the objects requiring accountability, confidentiality, and integrity. The audit policy imposes some structure on the nature of DBMS audit to ensure that the audit process is successful in obtaining its statement of goals. Further, the exercise of creating a policy is justified by the outcome of the audit

because it is more likely to be systematic, rather than random, and built on measures validated with risk management and quality-assurance policies.

7.0 LIST OF REFERENCES

- [AAP90] American Academy of Pediatrics, Committee on hospital Care, *Medical Staff Appointment and Delineation of Pediatric Privileges in Hospitals*, Pediatrics Vol. 85. April 1990, p.p. 607-617.
- [Dav87a] Davis, Richard, *FileMan: A Database Manager User's Manual*, Comptec, Washington D.C. 1987.
- [Dav90b] _____, *FileMan: A Database Manager User's Manual Volume II* Comptec, Washington D.C. 1990.
- [Bony81] Bonyun, D., *The Role of a Well Defined Auditing Process in the Enforcement of Privacy and Data Security*, Proceedings of the 1981 IEEE Symposium on Security and Privacy, April 1981, p.p. 19-25.
- [Clark87] Clark, David D. and David R. Wilson., *A Comparison of Commercial and Military Computer Security Policies*, Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 1987, p.p. 184-94.
- [DATE90] Date, C.J., *A Contribution to the Study of Database Integrity*, in Relational Database Writings 1985-1989, Addison-Wesley, 1990, p.p. 185-209.
- [DHCP91] Decentralized Hospital Computer Program, Department of Veterans Affairs, Veterans Health Administration, Medical Information Resources Management Office, 1991.
- [HAT89] Hospital Administration Terminology, American Hospital Association, Chicago, Illinois, 1989.
- [Karger88] Karger, Paul A., *Implementing Commercial Data Integrity with Secure Capabilities*, Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988, p.p. 130-39.
- [NCSC88a] National Computer Security Center (NCSC), *A Guide to Understanding AUDIT in Trusted Systems*, NCSC-TG-001 Version-2, 1 June 1988.
- [NCSC91b] _____, *Integrity in Automated Information Systems*, C Technical Report 79-91, September, 1991.
- [NCSC88c] _____, *Glossary of Computer Security Terms*, NCSC-TG-004, Version-1.21, October 1988.
- [NCSC91d] _____, *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, NCSC-TG-021, Version-1, April 1991.
- [Nixon90] Nixon, S.J., *Defining Essential Hospital Data*, British Medical Journal, vol. 300 February, 1990, p.p. 380-381.

- [Lich86] Lichtig, Leo K., *Hospital Information System for Case Mix Management*, John Wiley and Sons, 1986.
- [Mun86] Munnecke, Thomas H. and Ingeborg M. Kuhn, *Large Scale Portability of Hospital Information System Software*, IEEE 1986 Computer Applications In Medical Care, 1986.
- [Scha90] Schaefer, Marvin, et al, *Secure DBMS Auditor*, Rome Air Development Center (RADC), Griffiss Air Force Base, NY, RADC-TR-90-130, July 1990.
- [Seddon90] Seddon, Dr. J.T.S. quoted from New Zealand Medical Journal (1989; Vol 102; p.p 644-7) reported in British Medical Journal, vol. 300 February, 1990, p. 381.
- [TCSEC85] National Computer Security Center, *DOD Trusted Computer Systems Evaluation Criteria*, DOD 5200.28-STD, 1985.

On the Axiomatization of Security Policy: Some Tentative Observations About Logic Representation

James Bret Michael, Edgar H. Sibley, Richard F. Baum, and Fu Li

Department of Information and Software Systems Engineering, George Mason University,
4400 University Drive, Fairfax, Virginia 22030-4444

Abstract

Formalization of security policies using a logic programming paradigm starts with the representation of each security policy as an axiom. Theorems describing the properties of the resulting set of axioms can then be formally derived with the aid of an automated reasoning system. However, the ability to generate a proof is dependent in part on the structuring of the entities and relationships within and between security policies. Errors in structuring them can result in no indication (i.e., proof) when a conflict in axioms exists, or in an incorrect proof when in fact there is no conflict. To reduce errors, we consider the effect of using a model representation as the "front-end" to axiom formulation. In particular, we present a case study that compares a model-based approach to one with no pre-structuring. Both approaches axiomatize security policies as clauses in predicate calculus. The two differ in that the model-based approach begins with the definition of a structural model of the entities, mechanisms, and relationships contained in the security policies. This schema is then used to guide the axiomatization of the security policies. In the other approach, the security policies are axiomatized without the aid of a schema. Reasoning about the set of security policies is accomplished by posing queries in the form of theorems to an automated reasoning system. The schema-based approach appears to produce fewer structuring errors in formalizing security policy than the non-schema-based approach. However, the schema-based approach introduces more rigid rule formulation, and thus may not allow some valid queries.

1. INTRODUCTION

An organization responsible for protecting sensitive information from unauthorized access typically has a set of security policies represented as natural language statements in a security policy manual. In order to comply with these policies, members of an organization need to be able to understand and correctly interpret them. However there is usually some degree of imprecision in natural language representations of security policies. *Precision* in this context is defined as the degree to which a security policy or set of policies are free of ambiguities and

omissions. Imprecision can lead to misunderstanding and misinterpretation which, in turn, can result in violation of security policies.

Similarly, a secure computer system must be able to correctly enforce security policies. Articulation of policies using formal methods is one approach to developing precise representations upon which to automate the reasoning about and application of security policies. However, the degree of precision attained in representing them is dependent upon how well a formal method is applied. Imprecision can arise when the policy entities, mechanisms, and relationships are not explicitly modeled. Without such a model, omissions and ambiguity may not be readily discernible to a human or computer system.

We present a case study that compares a model-based approach with one not using a prior structure. Both approaches axiomatize security policies as clauses in predicate calculus. The model-based approach begins with a structural model of the entities, mechanisms, and relationships contained in the security policies. This schema is used to guide the axiomatization of the security policies. The other approach axiomatizes the policies without the aid of a schema. Our goal is to provide an initial demonstration that a schema-based approach produces fewer structuring errors in formalizing security policy than the non-schema-based approach. Our hypothesis is that the overall logic rule formulation is simplified in the model-based approach by capturing many of the rules in the structural model and formulating rules in terms of this model. This results in partitioning the rules into two sets: one for purely structural information, and the other for policy dynamics. The axiomatization of intentional policy (e.g., epistemic notions of knowledge and belief) is outside the scope of this paper.

One of the problems in applying formal methods according to Holt and deChampeaux [6] is that the supporting tools are often not integrated. The integration of a structuring mechanism, formal method, and theorem prover for modeling and analyzing policy is one step in this direction. Reasoning about the policies is accomplished by posing queries (stated as theorems) to the OTTER resolution-style theorem-proving program [7] for first-order logic with equality. The ability of the reasoning system to generate a proof is dependent in part on the structuring of the entities, mechanisms, and relationships within and between security policies. Errors can result in the erroneous generation of no proof or an incorrect proof: failing to find rule conflicts or not detecting existing conflicts. The schema-based model helps reduce these errors. The problem of verification also arises in using formal methods to prove program correctness [5]. However, by modeling a secure system at the policy level, many of the modeling errors that arise in proving program correctness are eliminated because policies are closer to the social processes that define a system, e.g., assumptions made by system users and developers are more visible at the policy level versus being embedded in algorithms and code. However, strict conformance to a schema concept may prevent the formulation of queries of potential importance. This is why we take a trial-and-error approach to policy formalization in this paper; i.e., the schema is just an aid for guiding the axiomatization of policy.

2. SECURITY POLICY EXAMPLE

This case study is based on a slightly modified version of a security policy example posed at the 1991 IFIP WG 11.3 Workshop on Database Security [9]. The example consists of a set of security policies regarding the access by employees and visitors to a secured building containing company-sensitive information. The policies are intended to be a reasonable facsimile of an extract, the Security Manual (SM), from a firm's Policy Manual. The security policies are listed in Figure 1.

2.1 Method for axiomatizing security policies

An informal experiment is presented as the basis for comparing the two approaches. The authors split into two pairs for the two approaches; the groups worked independently in formalizing the security policies.

The non-schema-based approach consists of first representing each policy statement as a clause in predicate calculus, and then adding any real-world knowledge as real-world axioms that are deemed necessary to complete the linkage between security manual axioms.

The schema-based approach begins with the development of a structural model of the policy structures (i.e., classes) and their corresponding relationships. This model was represented as an extended entity-relationship (EER) model [10]. An EER model is an entity-relationship model [3] with classification hierarchies and aggregation. The feasibility of using the EER formulation as a vehicle to guide the construction of logic models has been studied by Ackley et al. [1]. The security policies are then axiomatized in two broad sets: one based upon the structural information contained in the EER model, the other based on the policy requirements written in terms of the EER model.

2.2 Criterion for making comparisons

The two formalizations of the security policies are compared based upon their ability to support the OTTER theorem prover in answering the queries given in Figure 2. Query 0 is from the 1991 IFIP paper. Also considered are the restrictions imposed on query formulation by the model-based representation.

3. AXIOMATIZATION WITHOUT A SCHEMA

In the first model of policy formulation, all facts were represented as axioms. They were then input to a theorem prover to determine whether the set of statements showed internal consistency. Next they were debugged by augmentation and modification until a consistent set was achieved. Finally, questions or situations were formulated and presented to the theorem prover to prove that the statement produced a consistent condition, show that the question was valid within the frame of reference, or show that a fallacy existed in the interpretation.

SM 0.	The Security Department is responsible for administering and enforcing security policy. All the following policies are examples of this.
SM 1.	The Security Department issues security badges.
SM 2.	The Security Department provides guards.
SM 3.	The Security Department patrols the area.
SM 4.	New employees are issued employee badges on their first day of work.
SM 5.	Guards issue visitor badges at the door to visitors if: the visitor signs the security log, shows identification to the guard, and the visitor visits an employee.
SM 6.	Visitors must be escorted at all times by an employee.
SM 7.	People must show their badges while in the building.
SM 8.	Violation of security policy must be reported to the Security Department.
SM 9.	Loss of a badge is a security violation.
SM 10.	All security escorts are employees.
SM 11.	The security log is permanently retained at the entry station.
SM 12.	Acceptable forms of identification include a current driver's license, current state non-driver identification card, or current passport.
SM 13.	If a person is in the building and does not have a security badge and is observed by an employee, then that employee must escort the person to the security department.
SM 14.	Visitors must surrender their security badge to the security escort upon leaving the building.
SM 15.	Employees may take their security badge with them when exiting the building.
SM 16.	Upon separation of service, an employee must surrender his or her security badge to the security department.
SM 17.	Employees who lose their security badge must obtain a replacement for it.
SM 18.	Employee and visitor badges can only be obtained on regular business days from the security department.
SM 19.	Any employee who is wearing someone else's employee badge has committed a security violation and is to be escorted to the security department.
SM 20.	A visitor must not wear an employee badge.

Figure 1. Extract of a corporation's Security Manual

Query 0.	Is it possible that a person inside a secure building can be unescorted and have surrendered his or her visitor badge to a security guard?
Query 1.	Can an employee be inside the building and not have a badge?
Query 2.	Can a visitor be inside the building and not display a security badge and not cause a security violation?
Query 3.	Can a visitor in the building display one badge but not a second badge?
Query 4.	Must an employee have a security escort?
Query 5.	Can an employee lose a badge and not create a violation?
Query 6.	Can a visitor be inside the building having shown an expired identification at entry?
Query 7.	Can a visitor that has exited the building display a badge?
Query 8.	Can an employee be in the building before the first day of employment and cause a security violation?
Query 9.	Can a person obtain a badge from the security department on a holiday?
Query 10.	Can a visitor be in the building and only be visiting another visitor and not cause a security violation?
Query 11.	Can a visitor be in the building without having been issued a badge?
Query 12.	Can an employee leave the building with his or her badge?

Figure 2. Queries about the corporation's security policies

3.1 Formalization of security policies

First, the statements in the Security Manual had to be translated into axioms; at this time, it became obvious that real-world knowledge also had to be added as axioms. The *real-world knowledge* consists of those aspects of the environment that may be assumed to be understood by a typical employee and are normally included in a Policy Manual. The lack of their explicit inclusion contributes to ambiguities or unexpected need for a synonym dictionary or

thesaurus [8] to facilitate the construction of relationships between axioms. This was solved by the addition of the set of "formalizable" Real-World (RW) statements listed in Figure 3.

As an example, statement SM 2 **The Security Department provides guards** means that there is an entity that will be represented by the term `security_department` and an entity termed a `guard` with a predefined relationship that this department **provides** all the guards. This can then be represented as

$$\exists a \forall b \bullet (\text{security_department}(a) \wedge \text{guard}(b) \supset \text{provides}(a, b)).$$

Similarly, in order to deal with the fact that **Visitors are not employees** (RW 3), the encoding in Appendix 1 indicates that there is no way that an employee can be considered a visitor

$$\forall a \bullet \text{visitor}(a) \supset \neg \text{employee}(a).$$

RW 1.	The Acme Corporation has a secured building (termed "the building").
RW 2.	The Acme Corporation has a Security Department.
RW 3.	Visitors are not employees.
RW 4.	Employees are not visitors.
RW 5.	Employees and visitors are people.
RW 6.	Not all employees are security escorts, but guards are employees.
RW 7.	All employees are assigned to a department.
RW 8.	An employee may walk without a security escort in a secure building.
RW 9.	The regular business day is either Monday, Tuesday, Wednesday, Thursday, or Friday.
RW 10.	A security violation results from non-compliance with security policy.
RW 11.	A holiday is not a regular business day.
RW 12.	Expired identification (e.g., out of date) is not <i>current</i> .

Figure 3. Initial set of real-world facts

3.2 Initial analysis and modifications

After the real-world knowledge had been encoded, problems were detected regarding interpretation and cross referencing between one or more statements. This was an indication that the statements had to be further refined to correct errors. The initial set of axioms that were added is enumerated in Figures 4 and 5. The formalization of these axioms appears in Appendix 1. Some of the errors encountered during axiomatization included the following:

1. Statement SM 3 that **the Security Department patrols the area** needs a *demodulator* to express the fact that this area is in a secure building: this calls for addition of a "real world" fact, ARW 1 (Addition to the Real World).
2. Statement SM 4 says that New employees are issued an employee badge on their first day of work. This also needs a demodulator: ARW 2.
3. Statement SM 7 uses the term **display**, while statement SM 20 uses the term **wear**. This is a synonym or demodulator problem, solved in ARW 3.
4. The words **person** and **people** (see SM 7 and SM 13) must be equated, as in ARW 4.
5. SM 8 states that **Violation of security policy must be reported to the Security Department . . .** This is ambiguous; *someone* must be designated to do the reporting.

SM 8 is therefore modified to SM 8' (Any employee who observes a violation of security must . . .).

6. Who loses a badge in SM 9? ARW 5 was needed to fix this.
7. In SM 8 and 9, Security violation and violation of security must be equated; hence ARW 6.
8. SM 11 assumes there is only one entrance, hence ARW 7. There are also several synonyms: guard desk, door, entry station, and exit, which are equated in ARW 8.
9. There is a tacit assumption that SM 12 refers to a visitor; the revised version that clarifies this is SM 12': Acceptable forms of identification for a visitor include . . . Also, an acceptable form of identification must be stated to be equivalent to identification, as shown in ARW 9.
10. SM 13 is somewhat wordy (to ensure no ambiguities); this was to ensure no problems with its interpretation.
11. In SM 14, a security escort is any employee who is escorting a visitor. An addition (ASM) to the SM was made to this to improve "maintainability" of later versions of the manual; e.g., there may be a future requirement that restricts *who* can escort all or *certain types* of visitors. Hence ASM 1, which could be easily changed, as needed, for such changes to the Security Manual.
12. Leaving in SM 14 must be related to exiting in SM 15. ARW10 equates the terms.
13. SM 14 assumes that the visitor has not lost the badge at a time prior to exiting; this is also true for employees (in SM 16). This was, however, left as an example that shows the value of the theorem prover in highlighting linking and naming problems.
14. In SM 17, Losing a security badge means that a person no longer has it, as in ARW 11.
15. For SM 18, both employee and visitor badges must be defined as instances of a security badge. This is given in ARW 12.
16. SM 20 is stated *positively* in the formalization as a security violation (see also RW 10).

Naming and linking errors are typical of the generic problem of representing policy. As an example, suppose that the personnel department (which is within the secure building) adds the rule:

A new employee must sign in at the personnel office before going to the security department.

With the addition of this rule to the policy set, policies SM 18 and 20 and all policies related to them become inconsistent *unless* the "new employee" is treated initially as a "visitor." That is, an unsatisfiable condition exists until provision is made for an new employee to get into the building to obtain a badge.

SM 8'.	Any employee who observes a violation of security must report this violation to the Security Department
SM 12'.	Acceptable forms of identification for a visitor include a current driver's license, current state non-driver identification card, or current passport.
ASM 1.	A security escort is any employee.

Figure 4. Necessary modifications and additions to the security manual

ARW 1.	The area that is patrolled is secured.
ARW 2.	An employee badge is a type of security badge.
ARW 3.	A person displaying a badge is equivalent to a person wearing or showing it.
ARW 4.	Person is the singular version of the word people.
ARW 5.	People may lose their badge
ARW 6.	Security violation and violation of security are equivalent.
ARW 7.	There is only one entrance to the secure building.
ARW 8.	The words entry and exit have the following synonyms: guard desk, door, entry station.
ARW 9.	An acceptable form of identification is the same as identification.
ARW 10.	Leaving a building is equivalent to exiting a building.
ARW 11.	Losing a security badge is equivalent to the person no longer has a security badge.
ARW 12.	Both employee badge and visitor badge are both instances of security badge.

Figure 5. Additions to the real-world knowledge (ARW)

4. AXIOMATIZATION USING A SCHEMA

We now present a combined object-oriented and logic formulation approach to modeling the security manual policies and real-world constraints. The "structural" information was extracted from the general policy statements, and is represented as separate structural knowledge in a schema which consists of an object-oriented specification of class inclusion (e.g., *employee is_a [ISA] person*), class relationships (e.g. *employee [belong_to] dept*), and class properties or functions (e.g., *separated_from_service(employee)*). An additional component was the stipulation or definition of terms (e.g., *regular_business_day* is equivalent to *Monday, ..., Friday*). These components were then reformulated as logic statements. Finally, the remaining (non-structural) policy statements were written as logic statements in terms of the constructs in the structural model. The representations in the model and non-model approach are intended to be equivalent.

4.1 An object-oriented structural model

Overview of structural model

We first develop the object-oriented structural model. It is assumed that the reader is familiar with the fundamentals of object-oriented representation (e.g., see [2]). This model of the central entities in the Security Manual is shown in Figure 6. This model structure conforms to the EER model. It uses only single inheritance, represented by directed arcs. Relationships between classes are denoted by undirected arcs, with accompanying relationship names. An attribute of a class is denoted by '*', with the domains of the class attributes indicated. Attributes are represented in the model by properties (or functions) on the classes. For example, the statement *visitor (called v) signs the log* is represented as *signs_log(v)*, where *signs_log* is an attribute of visitor, but is now stated as a property.

The following relations are associated with the object *Security_Department* and are based on the Security Manual statements.

Administers(security_dept, security_policy),
 Enforces(security_dept, security_policy),
 Patrols(security_dept, area), and
 Provides(security_dept, guards)
 (subsumed by relation **Belong_to**(employee, dept)).

These relations are not directly used in defining dynamic behavior of the policy set. For example, SM 3 states: **The Security Department patrols the area**. SM 3 is not referenced by any other statement. For this reason the above relations are not shown in Figure 6. Similarly, statements RW1 and RW 2 are not directly referenced elsewhere and are also not depicted. The relations corresponding to RW 1 and RW 2 are as follows:

Belongs_to (secured_building, Acme_Corporation), and
 Unit_of (security_dept, Acme_Corporation).

Unlike a typical object-oriented model, class behavior within policy statements is not modeled here by methods and message passing. Instead, policy dynamics are represented by logic statements formulated in terms of classes, class relationships, and/or class attributes (properties). This is consistent with our desire to isolate class structures from the policy rules, and ultimately to express the model as logic statements. The translation of methods/messages adds an extra step to logic formulation and complicates the demarcation of class structure and policy actions, and is not addressed in this paper.

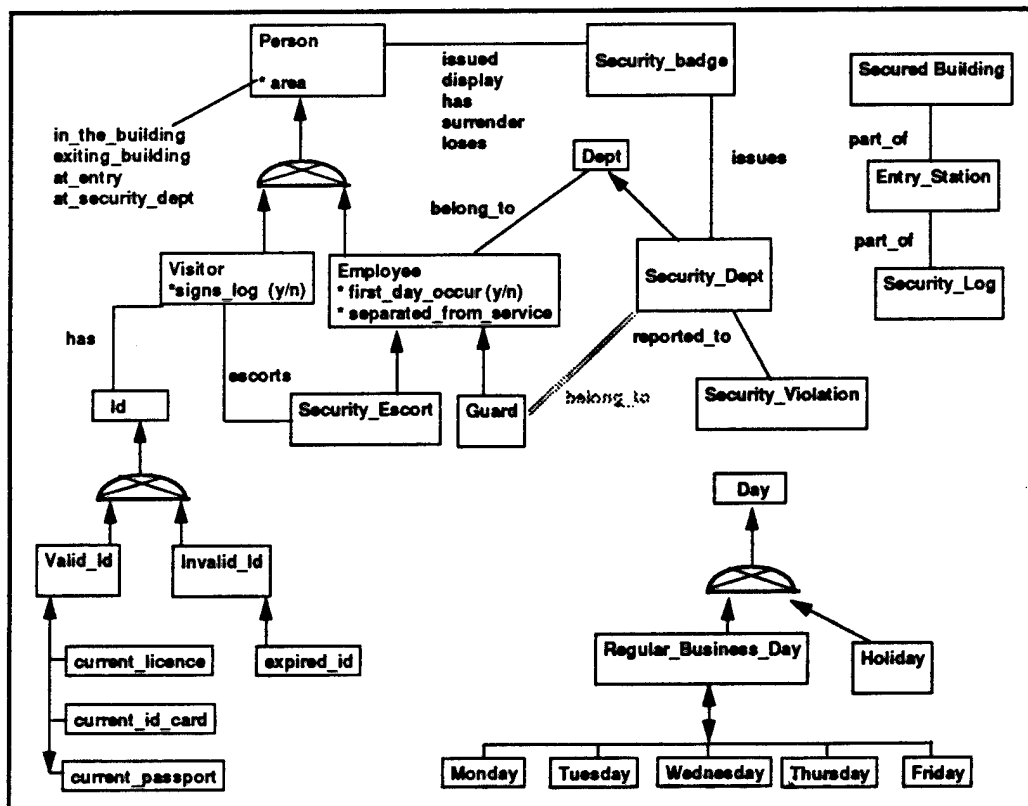


Figure 6. Object-oriented model of the security manual policy objects

Development of structural model

The structural model is developed in four steps. The first three yield an object-oriented class structure for the model. In the last, the isolated class methods are determined, along with the specification of terms made in the policy set.

The first step consists of identifying the model classes in the policy statements. Class determination is based on the analysis of noun/verb pairs. The policy statements directly used in class determination are as follows:

(part of) SM 2, SM 10, SM 11, (part of) SM 18, RW 3, RW 4, RW 5, RW 6, RW 7, and RW 11.

We found that policy statements SM 15, SM 20, and RW 8 are implied by the other rules of the structural model. Hence they are not explicitly stated. For example, the result of Query 4 (see Appendix 4) indicates that RW 8 is implied and consistent with the rule set, and therefore does not need to be included in the structural model.

In step two, the relationships between the model classes (class hierarchies and class relationships) are specified. Class attributes (e.g., **area** (location) of a **person**) are used to depict simple class properties. For instance, **in_the_building(person)** denotes that object **person** is *located within the secured building*. The class inheritance hierarchy is primarily based on the policy statements used to determine the classes. Next inter-class relationships are represented. To this end, the policy statements are decomposed into components using the actions (verbs) which link classes. The corresponding relationship is then expressed via the classes; e.g., policy RW 7 states that **All employees are assigned to a department**, expressed by the relationship **belong_to** between the class **employee** and the class **dept**. An alternative approach proposed by Dobson [4] would treat, for instance, a security guard as two or more separate classes: an escorting agent, a guarding agent, and so on.

Class instances (objects) specified by the policy statements are identified in step 3. These are determined from fairly explicit policy statements, such as RW 2: **The Acme Corporation has a Security Department**. The specific policy statements used are

RW 1, RW 2, (part of) RW 6.

The OTTER rule set of Appendix 2 contains a listing of these class objects in the section *Class Instances*.

In step four, we identify the definition or specification of various policy terms, especially those statements that specify permissible domain values such as RW 9: **The regular business day is either Monday, ..., Friday**. The policy statements used are:

SM 12 and RW 9.

Almost all of the allowable term values are treated via object inheritance; e.g., the day **Monday ISA** (is a subtype of) **regular_business_day**. Values for valid class properties (attributes) are similarly treated; e.g., **Area** can only consist of one of four domain values,

though this is not explicitly enforced in the current model. For this reason, values for area are not treated through inheritance and are only listed in Figure 6.

We also identify those class actions that are not explicitly used in other statements, such as SM 0: **The Security Department is responsible for enforcing . . . security policy.** These class actions are formulated as relations involving the class `security_dept`. The policy statements involved are

SM 0, SM 2, SM 3, RW 1, RW 2, and RW 10.

Since the above statements are distinct from the rest of the policy statements, they are included in the set of axioms as “passive rules.” As such, these rules are used by the theorem prover for subsuming other clauses while constructing a proof tree, but not for forward chaining. Given this class-object structure, we then formulate the necessary policy rule set as logic statements; see Appendix 2, *Class Structure*.

4.2 Non-Structural Portion of the Object-Based Approach

The second set of rules is a translation of the non-structural policy statements into logic statements. These statements are written in terms of the structural model constructs. The policy statements used are

SM 1, SM 4, SM 5, SM 6, SM 7, SM 8, SM 9, SM 13, SM 14, SM 16, SM 17, SM 18, SM19.

By encoding the rule set in terms of the structural model, we are forced to address the problems of cross-references, synonyms, and interpretation while formulating the rules, rather than after rule formulation (as occurred in Section 3). Hence there is less need for demodulation rules in this second rule set. On the other hand, we are required to rephrase various policy statements to explicitly refer to the constructs of the structural model *before* formulating the rules. For example, rule RW 12 states **Expired identification (e.g. out of date) is not current.** The term “current” has no direct counterpart in this model. The intended meaning, however, is that such an identification is not valid. An invalid identification is therefore *directly* represented and RW 12 is rewritten as RW 12": **Expired identification (e.g. out of date) is not valid identification.** Likewise, other policy statements are rewritten to refer directly to the structural model as depicted in Figure 7.

SM 3"	The Security Department patrols the building.
SM 5"	Note: No distinction is made between employee and visitor badges since this is a common attribute (there is no reason to differentiate).
SM 6"	Visitors must be escorted at all times by security escorts while in the building.
SM 18"	Security badges can only be obtained on regular business days from the security department.
SM 20"	Note: Meaningless in this representation, since there is only one type of security badge (implemented via inheritance).
RW 12"	Expired identification (e.g., out of date) is not valid.

Figure 7. Policy statements as revised for the structural model

After rephrasing the policy rules in terms of model constructs, vague or "non-connected" policy statements still remain. For example, to identify certain security violations, we must explicitly state that **if a person displays a badge, then that person must have the badge**. In addition, as previously stated, we need to add some real world assumptions to provide statement connections for making deductions about the policies. For example, **a security violation occurs if a person has two security badges** is added to the axiom set. The residual (or non-structural) statements of the Security Manual are then coded, and appear in Appendix (after the section *Class Structure*).

5. OBSERVATIONS

Before discussing the actual similarities in and differences between the two representations it is useful to consider the effect of the theorem prover.

5.1 Effects of the theorem prover

To understand how OTTER processes logic statements we found it was necessary to learn how OTTER parses axioms stated as formulas (i.e., with existential quantifiers and implication). This is now briefly discussed with respect to assertions and queries against the data set.

1. Consider the process of stating rule assertions with universal and existential quantifiers. For the statement **all visitors have a badge**, if we write this as:

$$(\text{all } x (\text{exists } y (\text{visitor}(x) \ \& \ \text{badge}(y) \ \& \ \text{has}(x,y)))) \quad (1)$$

OTTER will form three clauses, one being:

$$\text{visitor}(x) \quad (2)$$

that is,

$$(\text{all } x \text{ visitor}(x)) \quad (3)$$

If we add the statement **there exists an employee and employees are not visitors**, there is now a built-in contradiction, since OTTER will assert that for some constant d (denoting something exists—actually the Skolemization term)

$$\text{employee}(d) \quad (4.1)$$

$$(-\text{visitor}(d)) \quad (4.2)$$

yielding a contradiction between (2) and (4.2). That is, (2) states that everything, including the existing employee, must be a visitor, which contradicts (4.2).

However, if assertion (1) is written as an implication, this problem does not arise. For then assertion (1) becomes:

$$(\text{all } x (\text{visitor}(x) \rightarrow (\text{exists } y (\text{badge}(y) \ \& \ \text{has}(x,y)))))) \quad (5)$$

OTTER interprets (5) as possible alternatives

$$-\text{visitor}(x) \mid \text{badge}(c) \mid \text{has}(x,c) \quad (6)$$

where c is a constant. Now everything (including the existing employee) need *not* be a visitor. Hence, there is now no contradiction within OTTER between assertion (4) and (6), and thus the original two statements.

Therefore, when writing general rules which include existence statements, it is generally (but not always) necessary to use implication with universal quantification. That is, when stating general rules, it may be that the universal statement

$$(\text{all } x \text{ exists } y (P(x) \ \& \ Q(y))) \quad (7)$$

which results in the OTTER statements

$$P(x) \quad (8.1)$$

$$Q(c) \quad (8.2)$$

should be restated as the implication

$$(\text{all } x (P(x) \rightarrow (\text{exists } y Q(y)))) \quad (9)$$

which results in the OTTER representation

$$-P(x) \mid Q(c) \quad (10)$$

a statement of two distinct possibilities. In particular, no assertion is made about $P(x)$ always holding.

2. Queries can be used to test of the rule set. The same but “opposite” consideration must be used in writing queries. Here, you generally need to avoid implication when using OTTER. For example, to test **is it possible to have a visitor without a badge?**, it is better to state

$$(\text{exists } x (\text{all } y (\text{visitor}(x) \ \& \ \text{badge}(y) \ \& \ -\text{has}(x,y)))) \quad (11)$$

OTTER will turn this into disjoint clauses, will add them to the rules or set of support (sos), and will attempt to find contradictions. If, however, the query is phrased as:

$$(\text{exists } x ((\text{visitor}(x) \rightarrow (\text{all } y (\text{badge}(y) \ \& \ -\text{has}(x,y)))))) \quad (12)$$

this query may succeed (i.e., be found true), where the first one may not because OTTER will turn this statement into one statement of distinct alternatives

-visitor(c) | badge(y) | has(c,y) (13)

and this may be true "vacuously"—if it is possible to have no visitors, the statement above is true. Therefore when writing queries to test rules it is generally (but not always) better to avoid using implication if the statement involves existence. That is,

(exists x (P(x) -> (all y Q(y)))) (14)

may need to be rewritten as

(exists x all y (P(x) & Q(y))) (15)

The above observations are dependent on how OTTER reformulates rules during theorem proving. Similar behavior may occur for other theorem provers; the above indicates points to consider in such an analysis.

5.2 Observations on the two approaches

The two axiomatizations of the Security Manual policies are first compared based on the number and types of axioms that were generated. This information is summarized in Table 1.

Table 1
Comparison of statements produced in the two approaches

Type	Number of Clauses	
	Non-Schema-based Approach	Schema-based Approach
Constraints	21	13
Schema	NA	17
Real World	12	0
Implied by schema	NA	3
Added	63	8
Total	96	41
Total without schema	N/A	24

The total number of axioms required using the non-schema-based approach was more than twice that required for the schema-based approach, and four times as many if the schema (structural) axioms are not counted in the schema-based application. The reduction in the number of axioms in the object-based approach is primarily due to the embedding of many of the necessary axioms in the schema. In the theorem proving phase of the non-structured approach, naming and linkage errors required the addition of 48 ADTP (Added During Theorem Proving) axioms. The ADTP axioms represent real-world knowledge that was necessary to explicitly represent in order to correct missing linkages between policies and naming problems. The inability of the theorem prover to prove a theorem (answer a query) about the policy set indicated the existence of an error in the formulation of the theorem or in naming or

linkage. Most of this knowledge was captured during the first refinement in the schema-based model.

There are certain restraints imposed by the object-based approach. They may result in certain overly constraining rules that appear reasonable to a data administrator, but actually cause real-world problems. As an example in the stated approach, there is only one badge-type associated with a person. Thus the concept of a visitor accidentally or deliberately exchanging badges with an employee is essentially impossible due to internal constraints of the model. This is not to say that it would not be possible to define a correct representation of the policy, but rather that an overzealous data definer (administrator) could ignore the real world, because "It really shouldn't happen." In contrast, the unstructured approach resulted in the specification of many types of badges and the attendant problems of correctly naming and linking all of the different badge types.

The formalization approach taken also influenced the framing of queries about the set of policies listed in Figure 2. The schema-based approach resulted in fewer explicit linkages needing to be made within the queries. The queries and proofs appear in Appendices 3 and 4. The queries were run one at a time (and with no other background processes) on an IBM-compatible PC with a 20 MHz 80386 processor, 8 MB of RAM, and MS-DOS version 5.0. Both the hyperresolution and unit resolving (UR) resolution inferences rules were applied by the theorem prover. Some clause generation and resource usage statistics are presented in Tables 2 and 3. Note that the theorem prover generated a much smaller search space and consumed less resources (both in terms of memory usage and run time) when using the schema-based axiomatization. One reason for this outcome is that the schema-based approach provided the theorem prover with more compact and fewer axioms. The "passive" axioms were identifiable from the class structure (as disjoint nodes) and placed in the list of passive axioms so that OTTER would not try to use them in the search.

Table 2
Number and types of clauses generated by OTTER for the unstructured dataset

Clauses	Query												
	0	1	2	3	4	5	6	7	8	9	10	11	12
Input	0	0	0	0	0	0	0	0	0	0	0	0	0
given	16	8	3	20	13	4	26	21	7	41	23	17	27
generated	28	19	12	49	58	18	70	62	12	79	56	28	66
demod & eval rewrites	0	0	0	0	0	0	0	0	0	0	0	0	0
tautologies deleted	0	0	0	0	0	0	0	0	0	0	0	0	0
forward subsumed	12	13	5	29	41	11	46	46	2	43	33	11	37
subsumed by sos	4	6	5	13	19	10	22	21	1	18	18	5	21
kept	16	6	7	19	17	7	24	16	10	36	23	17	29
empty	1	1	1	1	1	1	1	0	1	1	1	1	1
back subsumed	49	3	26	48	9	1	20	0	38	23	22	86	32
sos size	0	0	0	0	0	0	0	0	0	0	0	0	0
Kbytes malloced	215	215	215	215	215	215	215	215	215	223	215	215	215
proof at (sec.)	8.52	7.47	6.98	9.18	8.13	7.03	9.72	9.67	7.31	10.77	9.18	8.35	9.72

Table 3
Number and types of clauses generated by OTTER for the structured dataset

Clauses	Query												
	0	1	2	3	4	5	6	7	8	9	10	11	12
Input	0	0	0	0	0	0	0	0	0	0	0	0	0
given	29	13	11	58	1	7	23	16	21	7	36	13	92
generated	93	27	23	337	2	17	39	41	54	19	173	32	17333
demod & eval rewrites	0	0	0	0	0	0	0	0	0	0	0	0	0
tautologies deleted	0	0	0	0	0	0	0	0	0	0	0	0	0
forward subsumed	54	12	10	221	0	8	12	16	24	10	126	15	17228
subsumed by sos	52	8	10	155	0	8	12	16	24	10	50	12	818
kept	38	15	13	116	2	9	27	25	30	9	47	17	105
empty	1	1	1	1	1	1	1	1	1	1	1	1	0
back subsumed	2	2	2	12	0	0	9	2	0	0	2	2	18
sos size	0	0	0	0	0	0	0	0	0	0	0	0	0
Kbytes malloced	135	119	119	151	119	119	127	167	127	119	135	119	167
proof at (sec.)	7.09	4.23	4.34	16.04	2.91	3.79	5.16	4.83	6.87	3.57	8.02	4.01	150.55

The formulation of the policies generated by the unstructured approach consisted of two categories of statements: roles and responsibilities, and actions. The need for creating links between these two categories of statements did not become evident until the theorem proving phase of the experiment. The theorem prover failed to prove any of the theorems until the linkages were made between these two categories of statements. For example, the responsibility of security guards to escort must_escort visitors had to be related to the actions escorted and escorted_by before many of the theorems could be proved. Some of the axioms only represented responsibilities or actions, but not both. As a result, there was initially no linkage between these axioms.

6. CONCLUSIONS

The recasting of security policies into an object-oriented model provides a basis for partitioning a set of policy statements into a structural and dynamic set of rules. The structural set represents the real-world relationships (schema) between entities; the dynamic set represents the policy actions themselves. The occurrence of naming and linkage errors can be reduced by using the schema to guide the axiomatization of real-world, passive, and procedural rules. In particular, the schema provides a common referent for terms used in the rules (similar to a data dictionary), and relations between the rules. For the unstructured approach, the number and complexity of axioms and the number of axiomatization errors increases as the axiom base is refined primarily because there is no model available to guide the structuring of the axioms or to facilitate the construction of connections between security policies.

The effects of this modeling approach upon such operations as the refinement and maintenance of security policy remain to be explored. For example, the imposition of a model can preclude posing unanticipated policy queries. This and determining heuristics for identifying under- and over-representation of policy are fertile areas for further research.

7. REFERENCES

1. Ackley, D., Carasik, R. P., Soon, T. S., Tryon, D. C., and Tsou, E.S., Tsur, S., and Zaniolo, C. Systems analysis for deductive database environments: An enhanced role for aggregate entities. *Proceedings of the 9th International Conference on the Entity-Relationship Approach*. COSMOPRESS, Geneva, Switzerland, 1991, pp. 129-142.
2. Booch, G. *Object Oriented Design with Applications*. Benjamin/Cummings Publishing, Redwood City, California, 1991.
3. Chen, P. P. The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems* 1 (1976), pp. 9-36.
4. Dobson, J. Information and denial of service. In C. E. Landwehr and S. Jajodia, eds., *Database Security, V: Status and Prospects*. Elsevier Science Publishers (North-Holland), Amsterdam, 1992, pp. 21-46.
5. Fetzer, J. H. Program verification: The very idea. *Communications of the ACM* 31 (1988), pp. 1048-1063.
6. Holt, R., and deChampeaux, D. A framework for using formal methods in object-oriented software development. *OOPS Messenger* 3 (April 1992), pp. 9-10.
7. McCune, W. B. *What's New in OTTER 2.2*. Technical Memorandum No. 153, Argonne National Laboratory, Argonne, Illinois, July 1991.
8. Sibley, E. H., Michael, J. B., and Wexelblat, R. L. An approach to formalizing policy management. In P. Bourguine and B. Walliser, eds., *Economics and Cognitive Science*. Pergamon Press, Oxford, England, 1992, pp. 155-169.
9. Sibley, E. H., Michael, J. B., and Wexelblat, R. L. Use of an experimental policy workbench: Description and preliminary results. In C. E. Landwehr and S. Jajodia, eds., *Database Security, V: Status and Prospects*. Elsevier Science Publishers (North-Holland), Amsterdam, 1992, pp. 47-76.
10. Teorey, T. J., Yang, D., and Fry, J. P (1986). A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys* 18 (1986), pp. 197-222.

Appendix 1.

Non-Object-Based Representation of Policy: Security Manual Policies

```

formula_list(usable).
%%% Security Manual policies
% AXIOM SM 0.
(all a (all b ((security_department(a) & security_policy(b)) -> (administers(a,b) & enforces(a,b))))).
% AXIOM SM 1. The security department issues security_badges.
(all a (all b ((security_department(a) & security_badge(b)) -> issues(a,b))))).
% AXIOM SM 2.
(all a (all b ((security_department(a) & guard(b)) -> provides(a,b))))).
% AXIOM SM 3.
(all a (all b ((security_department(a) & area(b)) -> patrols(a,b))))).
% AXIOM SM 4.
(all a (all b (all c ((new_employee(a) & first_day_of_work(b) & employee_badge(c))
-> (issued_to(a,b) & issued_on(c,b)))))).
% AXIOM SM 5.
(all a (all b (all c (all d (all e (all f (all g ((guard(a) & visitor(b) & visitor_badge(c) & door(d)
& identification(e) & employee(f) & security_log(g) & signs(b,g) & shown_by(e,b) & shown_to(e,a)
& visiting(b,f) & at(a,d) & at(b,d)) -> (issued_by(c,a) & issued_to(c,b) & issued_at(c,d)))))))))).
% AXIOM SM 6.
(all a (all b ((visitor(a) & employee(b)) -> (all c (all_times(c) & must_be_escorted_by(a,b)
& must_be_escorted_at(a,c)))))).
% AXIOM SM 7.
(all a (all b ((person(a) & security_badge(b)) -> (all c (building(c) & must_be_shown_by(b,a)
& must_be_shown_inside(b,c)))))).
% AXIOM SM 8.
(all a (security_policy_violation(a) -> (all b (security_department(b) & must_be_reported_to(a,b))))).
% AXIOM SM 9.
(all a (all b ((badge(a) & security_policy(b) & loss_of(a)) -> violation_of(b))))).
% AXIOM SM 10.
(all a (security_escort(a) -> employee(a))).
% AXIOM SM 11.
(all a (all b ((security_log(a) & entry_station(b)) -> permanently_retained_at(a,b))))).
% AXIOM SM 12.
(all a ((drivers_license(a) & current(a)) -> acceptable_form_of_identification(a))).
(all a ((state_non_driver_identification_card(a) & current(a)) -> acceptable_form_of_identification(a))).
(all a ((passport(a) & current(a)) -> acceptable_form_of_identification(a))).
% AXIOM SM 13.
(all a (all b (all c (all d (all e ((person(a) & building(b) & security_badge(c)
& employee(d) & security_department(e) & inside(a,b) & -has(a,c) & observed_by(a,d))
-> (must_escort(d,a) & must_be_escorted_to(a,e))))))))).
% AXIOM SM 14.
(all a (all b (all c (all d ((visitor(a) & security_badge(b) & security_escort(c)
& building(d) & leaving(a,d)) -> (must_surrender(a,b) & must_be_surrendered_to(b,c)))))).
% AXIOM SM 15.
(all a (all b (all c ((employee(a) & security_badge(b) & building(c) & exiting(a,c))
-> -must_surrender(a,b))))).
% AXIOM SM 16.
(all a (all b (all c (all d ((employee(a) & security_badge(b) & security_department(c)
& service(d) & separate(a,d)) -> (must_surrender(a,b) & must_be_surrendered_to(b,c)))))).
% AXIOM SM 17.
(all a (all b (all c ((employee(a) & security_badge(b) & replacement_security_badge(c) & lose(a,b))

```

```

-> must_obtain(a,c))))).
% AXIOM SM 18.
(all a (all b (all c (all d ((employee_badge(a) & visitor_badge(b)
& regular_business_day(c) & security_department(d)) -> (must_be_obtained_on(a,c) & must_be_obtained_on(b,c)
& must_be_obtained_from(a,d) & must_be_obtained_from(b,d)))))).
% AXIOM SM 19.
(all a (all b (all c (all d ((employee(a) & employee_badge(b) & security_violation(c)
& security_department(d) & -owned_by(b,a) & wearing(a,b))
-> (committed(a,c) & must_be_escorted_to(a,d)))))).
% AXIOM SM 20.
(all a (all b ((visitor(a) & employee_badge(b) & has(a,b)) -> -must_wear(a,b)))).
%%% Real-world constraints
% AXIOM RW 1. The Acme Corporation has a secured building.
(exists a (exists b ((Acme_Corporation(a) & secured_building(b)) -> has(a,b)))).
% AXIOM RW 2.
(exists a (exists b ((Acme_Corporation(a) & secured_department(b)) -> has(a,b)))).
% AXIOM RW 3. Visitors are not employees.
(all a (visitor(a) -> -employee(a))).
% AXIOM RW 4.
(all a (employee(a) -> -visitor(a))).
% AXIOM RW 5.
(all a ((employee(a) | visitor(a)) -> people(a))).
% AXIOM RW 6.
(exists a (employee(a) -> -security_escort(a))).
(all a (guard(a) -> employee(a))).
% AXIOM RW 7.
(all a (all b ((employee(a) & department(b)) -> assigned_to(a,b)))).
% AXIOM RW 8.
(all a (all b (all c ((employee(a) & secure_building(b) & inside(a,b))
-> (security_escort(c) & -escorted_by(a,c)))))).
% AXIOM RW 9.
(all a ((Monday(a) | Tuesday(a) | Wednesday(a) | Thursday(a) | Friday(a)) -> regular_business_day(a))).
% AXIOM RW 10.
(all a (all b (all c ((security_violation(a) & employee(b) & security_policy(c)
& -in_compliance_with(c,b)) -> committed_by(a,c))))).
(all a (all b (all c ((security_violation(a) & visitor(b) & security_policy(c)
& -in_compliance_with(c,b)) -> committed_by(a,c))))).
% AXIOM RW 11.
(all a (holiday(a) -> -regular_business_day(a))).
(all a (regular_business_day(a) -> -holiday(a))).
% AXIOM RW 12.
(all a (expired_identification(a) -> -current_identification(a))).
(all a (current_identification(a) -> -expired_identification(a))).
%%% Necessary additions for completing the linkages between SMs and RWs
% AXIOM SM 8'.
(all a (all b (all c ((employee(a) & security_violation(b) & security_department(c)
& observed(a,b)) -> (must_report(a,b) & must_be_reported_to(b,c)))))).
% AXIOM SM 12'.
(all a ((drivers_license(a) & current(a)) -> (all b (acceptable_form_of_identification(a) & visitor(b)
& for(a,b))))).
(all a ((state_non_drivers_license(a) & current(a)) -> (all b (acceptable_form_of_identification(a) & visitor(b) & for(a,b))))).
(all a ((passport(a) & current(a)) -> (all b (acceptable_form_of_identification(a) & visitor(b)
& for(a,b))))).
% AXIOM ASM 1.

```

```

(all a (security_escort(a) -> employee(a))).
% AXIOM ARW 1.
(all a ((area(a) & patrolled(a)) -> secured(a))).
% AXIOM ARW 2.
(all a (employee_badge(a) -> security_badge(a))).
% AXIOM ARW 3.
(all a (all b ((person(a) & badge(b) & displaying(a,b)) -> (wearing(a,b) | showing(a,b))))).
% AXIOM ARW 4.
(all a (person(a) -> people(a))).
(all a (people(a) -> person(a))).
% AXIOM ARW 5.
(all a (all b ((person(a) & security_badge(b)) -> (lose(a,b) | -lose(a,b))))).
% AXIOM ARW 6.
(all a (security_violation(a) -> violation_of_security(a))).
(all a (violation_of_security(a) -> security_violation(a))).
% AXIOM ARW 7.
(all a (secure_building(a) -> (exists b (single_entrance(a) & has(a,b))))).
% AXIOM ARW 8.
(all a ((guard_desk(a) | door(a) | entry_station(a)) -> entry(a))).
(all a ((guard_desk(a) | door(a) | entry_station(a)) -> exit(a))).
% AXIOM ARW 9.
(all a (acceptable_form_of_identification(a) -> identification(a))).
% AXIOM ARW 10.
(all a (all b ((person(a) & building(b) & leaving(a,b)) -> exiting(a,b))))).
(all a (all b ((person(a) & building(b) & exiting(a,b)) -> leaving(a,b))))).
% AXIOM ARW 11.
(all a (all b ((person(a) & security_badge(b) & lose(a,b)) -> -has(a,b))))).
% AXIOM ARW 12.
(all a (employee_badge(a) -> security_badge(a))).
(all a (visitor_badge(a) -> security_badge(a))).
(all a (employee_badge(a) -> -visitor_badge(a))).
(all a (visitor_badge(a) -> -employee_badge(a))).
%%% Axioms added during the theorem proving (ADTP).
% AXIOM ADTP 1.
(all a (secured_building(a) -> secure_building(a))).
% AXIOM ADTP 2.
(all a (secure_building(a) -> secured_building(a))).
% AXIOM ADTP 3.
(all a (all b ((visitor(a) & visitor_badge(b) & lose(a,b)) -> -has(a,b))))).
% AXIOM ADTP 4.
(all a (all b ((employee(a) & employee_badge(b) & lose(a,b)) -> -has(a,b))))).
% AXIOM ADTP 5.
(all a (all b ((visitor(a) & visitor_badge(b) & -has(a,b) & secure_buidling(c) & inside(a,c)) -> lose(a,b))))).
% AXIOM ADTP 6.
(all a (all b ((employee(a) & employee_badge(b) & -has(a,b) & secure_buidling(c) & inside(a,c))
-> lose(a,b))))).
% AXIOM ADTP 7.
(all a (all b (exists c ((person(a) & (visitor_badge(b) | employee_badge(b)) &
lose(a,b)) -> security_violation(c))))).
% AXIOM ADTP 8.
(all a (guard(a) -> security_escort(a))).
% AXIOM ADTP 9.
(all a (security_escort(a) -> guard(a))).
% AXIOM ADTP 10.

```

```

(all a (all b (all c (all d (all e ((guard(a) & visitor(b) & visitor_badge(c) & door(d)
& -acceptable_form_of_identification(e) & shown_by(e,b) & shown_to(e,a) & at(a,d) & at(b,d)
-> -issued_to(c,b))))))).
% AXIOM ADTP 11.
(all a (all b (all c ((visitor(a) & visitor_badge(b) & -issued_to(b,a) & secured_buidling(c)) -> -inside(a,c)))).
% AXIOM ADTP 12.
(all a (all b ((visitor(a) & visitor_badge(b) & -issued_to(b,a)) -> -has(a,b)))).
% AXIOM ADTP 13.
(all a (all b ((visitor(a) & visitor_badge(b) & -has(a,b)) -> (-issued_to(b,a) | lose(a,b)))).
% AXIOM ADTP 14.
(all a (door(a) -> single_entrance(a))).
% AXIOM ADTP 15.
(all a (door(a) -> entry(a))).
% AXIOM ADTP 16.
(all a (door(a) -> exit(a))).
% AXIOM ADTP 17.
(all a (single_entrance(a) -> door(a))).
% AXIOM ADTP 18.
(all a (single_entrance(a) -> entry(a))).
% AXIOM ADTP 19.
(all a (single_entrance(a) -> exit(a))).
% AXIOM ADTP 20.
(all a (entry(a) -> exit(a))).
% AXIOM ADTP 21.
(all a (entry(a) -> door(a))).
% AXIOM ADTP 22.
(all a (entry(a) -> single_entrance(a))).
% AXIOM ADTP 23.
(all a (exit(a) -> entry(a))).
% AXIOM ADTP 24.
(all a (building(a) -> secured_building(a))).
% AXIOM ADTP 25.
(all a (secured_building(a) -> building(a))).
% AXIOM ADTP 26.
(all a (all b (all c ((person(a) & secured_building(b) & exiting(a,b) & (visitor_badge(c)
| employee_badge(c)) & has(a,c)) -> (leaving(a,b) & must_surrender(a,c)))).
% AXIOM ADTP 27.
(all a (all b (all c ((person(a) & secured_building(b) & leaving(a,b)
& (visitor_badge(c) | employee_badge(c)) & has(a,c)) -> (exiting(a,b) & must_surrender(a,c)))).
% AXIOM ADTP 28.
(all a (all b (all c ((person(a) & (visitor_badge(b) | employee_badge(b))
& secured_building(c) & must_surrender(a,b)) -> (exiting(a,c) | leaving(a,c)))).
% AXIOM ADTP 29.
(all a (security_badge(a) -> (visitor_badge(a) | employee_badge(a)))).
% AXIOM ADTP 30.
(all a ((visitor_badge(a) | employee_badge(a)) -> security_badge(a))).
% AXIOM ADTP 31.
(all a (all b (all c (all d ((person(a) & (visitor_badge(b) | employee_badge(b))
& security_department(c) & holiday(d)) -> (-obtain(a,b) & -obtain_from(b,c)))).
% AXIOM ADTP 32.
(all a (all b (all c (all d ((person(a) & secured_building(b) & visitor_badge(d) &
-escorted_by(a,c) & surrender(a,d) & inside(a,b)) -> (exists e security_violation(e)))).
% AXIOM ADTP 33.
(all a (all b (all c ((person(a) & secured_building(b) & visitor_badge(c) &

```

```

surrender(a,c)) -> (-must_be_shown_by(b,d) & -must_be_shown_inside(c,b)))).
% AXIOM ADTP 34.
(all a (expired_identification(a) -> -acceptable_form_of_identification(a))).
% AXIOM ADTP 35.
(all a (-acceptable_form_of_identification(a) -> expired_identification(a))).
% AXIOM ADTP 36.
(all a (all b (all c ((person(a) & entry(b) & secure_building(c) & at(a,b)) -> -inside(a,c)))).
% AXIOM ADTP 37.
(all a (all b (all c ((person(a) & entry(b) & secure_building(c) & inside(a,c)) -> -at(a,b)))).
% AXIOM ADTP 38.
(all a (all b ((person(a) & badge(b) & display(a,b)) -> has(a,b)))).
% AXIOM ADTP 39.
(all a (all b ((person(a) & building(b) & exited(a,b)) -> -inside(a,b)))).
% AXIOM ADTP 40.
(all a (all b ((person(a) & building(b) & -inside(a,b)) -> exited(a,b)))).
% AXIOM ADTP 41.
(all a (all b (all c ((visitor(a) & visitor_badge(b) & building(c) & -inside(a,c)) -> -has(a,b)))).
% AXIOM ADTP 42.
(all a ((employee(a) | visitor(a)) -> person(a))).
% AXIOM ADTP 43.
(all a (person(a) -> (employee(a) | visitor(a)))).
% AXIOM ADTP 44.
(all a (all b (all c ((visitor(a) & visitor_badge(b) & building(c) & -inside(a,c)) -> -display(a,b)))).
% AXIOM ADTP 45.
(all a (all b (all c ((visitor(a) & visitor_badge(b) & door(c) & must_surrender(a,b))
-> (at(a,c) & surrender(a,b)))))).
% AXIOM ADTP 46.
(all a (all b ((visitor(a) & visitor_badge(b) & surrender(a,b)) -> -has(a,b)))).
% AXIOM ADTP 47.
(all a (all b ((person(a) & badge(b) & surrender(a,b)) -> (-display(a,b) & -has(a,b)))).
% AXIOM ADTP 48.
(all a (all b (all c ((visitor(a) & person(b) & building(c) & visiting(a,b) & inside(a,c) & inside(b,c))
-> employee(b)))))).

end_of_list.

```

Appendix 2.

Object-Based Representation of Policy: The Object Schema

```

formula_list(usable).
%%% policy statements which are covered by structure model
%%% Class structure
% RW 5.
(all x (employee(x) -> person(x))).      % part 1
(all x (visitor(x) -> person(x))).        % part 2
% SM 10.
(all x (security_escort(x) -> employee(x))).
% RW 6 (part 2): ...but guards are employees
(all x (guard(x) -> employee(x))).
% RW 3: Visitors are not employees.
(all x (visitor(x) -> -employee(x))).
% RW 4: Employees are not visitors.
(all x (employee(x) -> -visitor(x))).
% Implied real world assumptions

```

```

(all x (security_dept(x) -> dept(x))).
(all x (valid_id(x) -> id(x))).
(all x (invalid_id(x) -> id(x))).
(all x (valid_id(x) -> -invalid_id(x))).
(all x (invalid_id(x) -> -valid_id(x))).
(all x (holiday(x) -> day(x))).
(all x (regular_business_day(x) -> day(x))).
% RW 11: A holiday is not a regular business day.
(all x (holiday(x) -> -regular_business_day(x))). % if
(all x (regular_business_day(x) -> -holiday(x))). % only if
%%% Class instance (object)
% RW 6 (part 1): Not all employees are security escorts...
(exists x (employee(x) & -security_escort(x))).
% Implied by RW 1: There exists a secured building
% (termed "the building").
(exists x secured_building(x)).
% Implied by RW 2: There exists a security department.
(exists x security_dept(x)).
% The security department is unique.
% (implied by SM 0, SM 1)
(all x all y ((security_dept(x) & security_dept(y)) -> (x = y))).
%%% Class relationships
% RW 7: All employees are assigned to a department.
(all x exists y ((employee(x) & dept(y)) -> belong_to(x,y))).
% Implied by SM 2: The security department provides guards.
(all x exists y ((guard(x) & security_dept(y)) -> belong_to(x,y))).
% A secured building has an entry station.
% (implied by SM 5, SM 11)
(all x (secured_building(x) -> (exists y (entry_station(y) & part_of(y,x))))).
% SM 11: The security log is permanently retained at the entry station.
(all x (security_log(x) -> (exists y (entry_station(y) & part_of(x,y))))). % location of log
(all x all y all z ((security_log(x) & entry_station(y) & entry_station(z) % permanent log
& entry_station(y) & entry_station(z) & part_of(x,y) & part_of(x,z)) -> (y = z))).
%%% Definition of terms
% SM 12.
(all x (valid_id(x) -> (current_licence(x) | current_id_card(x) | current_passport(x)))). % if
(all x ((current_licence(x) | current_id_card(x) | current_passport(x)) -> valid_id(x))). % only if
% Implied by RW 12.
(all x (expired_id(x) -> invalid_id(x))).
% RW 9.
(all x (regular_business_day(x) -> (monday(x) | tuesday(x) | wednesday(x)
| thursday(x) | friday(x)))). % if
(all x ((monday(x) | tuesday(x) | wednesday(x) | thursday(x) | friday(x))
-> regular_business_day(x))). % only if
%%% functional connections, implications
% Person must have a badge to display it.
(all x all y ((person(x) & security_badge(y) & display(x,y)) -> has(x,y))).
% If a person has a badge but it was not issued, that is a violation.
(all x all y ((person(x) & security_badge(y) & -issued(x,y) & has(x,y)) ->
(exists z security_violation(z)))).
% If a person is issued a badge, it was done by the security department.
(all x all y ((person(x) & security_badge(y) & issued(x,y))
-> (exists z (security_department(z) & issues(z,y))))).
% Having 2 badges (more than one) is a violation.

```

```

(all x all y all z ((person(x) & security_badge(y) & security_badge(z) & has(x,y)
& has(x,z) & -(y = z)) -> (exists w security_violation(w)))).
% A person can be issued only one badge
(all x all y all z ((person(x) & security_badge(y) & security_badge(z) & issued(x,y)
& issued(x,z)) -> (y = z))).
% A person can not display a surrendered badge.
(all x all y ((person(x) & security_badge(y) & surrender(x,y)) -> -displays(x,y))).
% A person can not display a lost badge.
(all x all y ((person(x) & security_badge(y) & loses(x,y)) -> -display(x,y))).
% A person must obtain badge from security dept
% on a regular business day.
(all x all y all u ((person(x) & security_badge(y) & security_dept(u) & obtain_from(x,y))
-> (exists z (day(z) & obtain_badge_from_on(x,y,u,z)))).
%% the rest of the security manual represented as axioms
% SM 1.
(all x (security_badge(x) -> (exists y (security_dept(y) & issues(y,x)))).
% SM 4.
(all x all y ((employee(x) & security_badge(y) & issued(x,y)) -> first_day_occur(x))).
% SM 5.
(all w all x ((visitor(w) & security_badge(x) & issued(w,x)) ->
(exists z exists y exists t (at_entry(w) & guard(t) & at_entry(t)
& valid_id(y) & has(w,y) & shows_id_to(w,y,t) & employee(z) & visiting(w,z) & signs_log(w)))).
% SM 6.
(all x ((visitor(x) & in_the_building(x)) -> (exists y (security_escort(y) & escorts(y,x)))).
% SM 7.
(all x ((person(x) & in_the_building(x)) -> (exists y (security_badge(y) & display(x,y)))).
% SM 8.
(all x (security_violation(x) -> (exists y (security_dept(y) & report_to(x,y)))).
% SM 9.
(all x all y ((person(x) & security_badge(y) & loses(x,y)) -> (exists z security_violation(z)))).
% SM 13.
(all v all x exists y exists z ((person(v) & in_the_building(v) & security_badge(x)
& employee(y) & security_dept(z) & -display(v,x) & is_observed_by(v,y))
-> (escorts(y,v) & take_person_to(y,v,z) & at_security_dept(v))).
% SM 14.
(all w all x ((visitor(w) & exiting_the_building(w) & security_badge(x) & has(w,x)) ->
(exists y (security_escort(y) & escorts(y,w) & surrender_id_to(w,x,y) & surrender(w,x)))).
% SM 16.
(all w all x exists y ((employee(w) & security_badge(x) & security_dept(y)
& separate_from_service(w) & has(w,x)) -> (surrender_id_to(w,x,y) & surrender(w,x))).
% SM 17.
(all x all y exists z ((employee(x) & security_badge(y) & security_badge(z) & loses(x,y))
-> issued(x,z))).
% SM 18.
(all x all w all z all u (((person(x) & security_badge(w) & security_dept(z) & day(u)
& obtain_badge_from_on(x,w,z,u)) -> (regular_business_day(u) & issued(x,w)))).
% SM 19.
(all w all u all x ((employee(w) & employee(u) & security_badge(x)
& has(u,x) & display(w,x)) -> (exists y exists z exists t (security_violation(y) & security_dept(z)
& person(t) & take_person_to(t,w,z) & at_security_dept(w)))).
end_of_list.
%*****
% Formulas not used, but included for completeness
%*****

```



```

%%% Not directly used in other rules.
formula_list(passive).
% Acme Corp (AC) exists (implied).
acme_corp(AC).
% RW 1: The Acme Corporation has a secured building (termed "the building").
(exists y (secured_building(y) & belongs_to(y,AC))).
% RW 2: The Acme Corporation has a security department.
(exists y (security_dept(y) & unit_of(y,AC))).
%%% Class methods (listings)
% SM 0.
(all x (security_policy(x) -> (exists y (security_dept(y) & administers(y,x)))). % part 1
(all x (security_policy(x) -> (exists y (security_dept(y) & enforces(y,x)))). % part 2
% SM 2.
(all x (guard(x) -> (exists y (security_dept(y) & provides(y,x)))).
% SM 3.
(all x (secured_building(x) -> (exists y (security_dept(y) & patrols(y,x)))).
end_of_list.

```

Appendix 3. Queries Used in the Non-Schema Approach

```

% Query 0 (from 1991 IFIP paper)
(exists a (all b (all c (all d (person(a) &
secured_building(b) & security_guard(c)
& visitor_badge(d) & -inside(a,b)
& -escorted_by(a,c) & surrender(a,d)))).
----- PROOF -----
16 [] -person(x22) | -security_badge(x23)
| must_be_shown_inside(x23,x24).
135 [] -visitor_badge(x166)
| security_badge(x166).
143 [] -person(x175)
| -secured_building(x176)
| -visitor_badge(x177)
| -surrender(x175,x177)
| -must_be_shown_inside(x177,x176).
162 [] person($c6).
163 [] secured_building(x210).
165 [] visitor_badge(x212).
168 [] surrender($c6,x212).
173 [hyper,165,135] security_badge(x).
178 [hyper,173,16,162] must_be_shown_inside(x,y).
184 [ur,168,143,162,163,165]
-must_be_shown_inside(x,y).
185 [binary,184,178].

```

```

% Query 1.
(exists x all y all z (employee(x) & secured_building(y) &
inside(x,y) & badge(z) & -has(x,z))).
----- PROOF -----
83 [] -secure_building(x104)
| has(x104,$f1(x104)).
98 [] -secured_building(x118)
| secure_building(x118).
163 [] secured_building(y).

```

```

166 [] -has($c6,z).
171 [hyper,163,98] secure_building(x).
172 [hyper,171,83] has(x,$f1(x)).
173 [binary,172,166].

```

```

% Query 2.
(exists x all y all z (visitor(x) & visitor_badge(y) & -
has(x,y) & secure_building(z) & inside(x,z))).
Level of proof is 1, length is 1.
----- PROOF -----

```

```

83 [] -secure_building(x104)
| has(x104,$f1(x104)).
164 [] -has($c6,y).
165 [] secure_building(z).
173 [hyper,165,83] has(x,$f1(x)).
174 [binary,173,164].

```

```

% Query 3.
(exists x all y all z all w (visitor(x) & secured_building(w)
& inside(x,w)
& visitor_badge(y) & visitor_badge(z) & display(x,y) & -
display(x,z))).
----- PROOF -----

```

```

167 [] display($c6,y).
168 [] -display($c6,z).
188 [hyper,168,167].

```

```

% Query 4.
(exists a (exists b ((employee(a) & security_escort(b)) ->
must_be_escorted_by(a,b)))).
(employee(a) & secure_building(b) & inside(a,b)).
(all y (security_escort(y) ->
-must_be_escorted_by(a,y))).
----- PROOF -----

```

```

46 [] -employee($c5)
| -security_escort($c5).

```

```

49 [] -employee(x70)
| -secure_building(x71) | -inside(x70,x71)
| security_escort(x72).
73 [] -security_escort(x93) | employee(x93).
163 [] employee(a).
164 [] secure_building(b).
165 [] inside(a,b).
177 [hyper,165,49,163,164] security_escort(x).
182 [hyper,177,73] employee(x).
183 [ur,177,46] -employee($c5).
184 [binary,183,182] .

```

% Query 5.

```

(exists x exists y all z (employee(x)
& employee_badge(y) & lose(x,y)
& -security_violation(z))).

```

```

----- PROOF -----
105 [] -person(x128)
| -employee_badge(x129)
| -lose(x128,x129)
| security_violation($f2(x128,x129)).
152 [] -employee(x195) | person(x195).
162 [] employee($c7).
163 [] employee_badge($c6).
164 [] lose($c7,$c6).
165 [] -security_violation(z).
166 [hyper,162,152] person($c7).
172 [ur,166,105,163,165] -lose($c7,$c6).
173 [binary,172,164] .
----- end of proof -----

```

% Query 6.

```

(exists a (exists b (all c (all d (visitor(a) &
security_escort(b) & expired_identification(c) & has(a,c)
& shown_to(c,b) & entry(d) & at(a,d) & at(b,d) &
building(d) & inside(a,d)))))).

```

```

----- PROOF -----
98 [] -secured_building(x118)
| secure_building(x118).
122 [] -building(x154)
| secured_building(x154).
147 [] -person(x183) | -entry(x184)
| -secure_building(x185)
| -inside(x183,x185) | -at(x183,x184).
153 [] -visitor(x195) | person(x195).
162 [] visitor($c7).
167 [] entry(x211).
168 [] at($c7,x211).
170 [] building(x211).
171 [] inside($c7,x211).
172 [hyper,162,153] person($c7).
182 [hyper,170,122] secured_building(x).
191 [hyper,182,98] secure_building(x).
195 [ur,168,147,172,167,191]
-inside($c7,x).

```

196 [binary,195,171] .

% Query 7. Note: no conflict as expected.

```

% A visitor can display a badge outside,
% but this is a security violation; set of
% support is empty as expected.
(exists a (exists b (exists c (visitor(a) & building(b) &
exited(a,b) & badge(c) & display(a,c)))))).

```

% Query 8.

```

-(exists a (exists b (exists c (exists d ((employee(a) &
secure_building(b)
& -first_day_of_employment(c) & inside(a,b)) ->
security_violation(d)))))).

```

----- PROOF -----

```

46 [] -employee($c5)
| -security_escort($c5).
49 [] -employee(x70)
| -secure_building(x71) | -inside(x70,x71)
| security_escort(x72).
162 [] employee(x210).
163 [] secure_building(x211).
165 [] inside(x210,x211).
169 [ur,162,46] -security_escort($c5).
176 [ur,169,49,162,163] -inside(x,y).
177 [binary,176,165] .

```

% Query 9.

```

(exists x exists y exists z exists u exists w (person(x) &
visitor_badge(y) & employee_badge(w)
& security_department(z) & ((obtain(x,y) &
obtain_from(x,z) | (obtain(x,w) & obtain_from(x,z)))
& holiday(u))).

```

----- PROOF -----

```

137 [] -person(x167) | -visitor_badge(x168) | -
security_department(x169)
| -holiday(x170) | -obtain(x167,x168).
139 [] -person(x167)
| -employee_badge(x168)
| -security_department(x169)
| -holiday(x170) | -obtain(x167,x168).
162 [] person($c10).
163 [] visitor_badge($c9).
164 [] employee_badge($c6).
165 [] security_department($c8).
166 [] obtain($c10,$c9) | obtain($c10,$c6).
170 [] holiday($c7).
178 [ur,170,139,162,164,165]
-obtain($c10,$c6).
180 [ur,170,137,162,163,165]
-obtain($c10,$c9).
206 [hyper,166,180] obtain($c10,$c6).
207 [binary,206,178] .

```

% Query 10.

(exists a (exists b (all c (all d (visitor(a) & visitor(b) & building(c) & inside(a,c) & inside(b,c) & visiting(a,b) & -security_violation(d)))))).

----- PROOF -----

```

43 [] -employee(x65) | -visitor(x65).
153 [] -visitor(x195) | person(x195).
161 [] -visitor(x207) | -person(x208)
| -building(x209) | -visiting(x207,x208)
| -inside(x207,x209)
| -inside(x208,x209) | employee(x208).
162 [] visitor($c7).
163 [] visitor($c6).
164 [] building(x210).
165 [] inside($c7,x210).
166 [] inside($c6,x210).
167 [] visiting($c7,$c6).
172 [hyper,163,153] person($c6).
174 [ur,163,43] -employee($c6).
191 [ur,166,161,162,172,164,165,174]
-visiting($c7,$c6).
192 [binary,191,167] .

```

% Query 11.

(all x (all y (all z (visitor(x) & visitor_badge(y) & -issued_to(x,y) & secure_building(z) & inside(x,z))))).

Level of proof is 1, length is 2.

----- PROOF -----

```

83 [] -secure_building(x104)
| has(x104,$f1(x104)).
110 [] -visitor(x140) | -visitor_badge(x141)
| issued_to(x141,x140) | -has(x140,x141).
162 [] visitor(x).
163 [] visitor_badge(y).
164 [] -issued_to(x,y).
165 [] secure_building(z).
173 [hyper,165,83] has(x,$f1(x)).
183 [ur,164,110,162,163] -has(x,y).
184 [binary,183,173] .

```

% Query 12. Note: no conflict as expected.

(employee(a) & building(b) & exiting(a,b)).
(security_badge(b) & has(a,b)
& -surrender(a,b)).

----- PROOF -----

```

14 [] -person(x22) | -security_badge(x23)
| building(x24).
29 [] -employee(x44)
| -security_badge(x45) | -building(x46)
| -exiting(x44,x46)
| -must_surrender(x44,x45).
92 [] -person(x110) | -building(x111)
| -exiting(x110,x111) | leaving(x110,x111).
122 [] -building(x154)
| secured_building(x154).
129 [] -person(x159)
| -secured_building(x160)
| -leaving(x159,x160) | -visitor_badge(x161) | -
has(x159,x161)
| must_surrender(x159,x161).
131 [] -person(x159)
| -secured_building(x160)
| -leaving(x159,x160)
| -employee_badge(x161)
| -has(x159,x161)
| must_surrender(x159,x161).
134 [] -security_badge(x165)
| visitor_badge(x165)
| employee_badge(x165).
152 [] -employee(x195) | person(x195).
162 [] employee(a).
164 [] exiting(a,b).
165 [] security_badge(b).
166 [] has(a,b).
168 [hyper,162,152] person(a).
176 [hyper,168,14,165] building(x).
179 [hyper,176,122] secured_building(x).
191 [hyper,164,92,168,176] leaving(a,b).
192 [ur,164,29,162,165,176]
-must_surrender(a,b).
194 [ur,192,131,168,179,191,166]
-employee_badge(b).
195 [ur,192,129,168,179,191,166]
-visitor_badge(b).
196 [ur,194,134,165] visitor_badge(b).
197 [binary,196,195] .

```

Appendix 4.

Queries Used in the Object-Based Approach

% Query 0 (from 1991 IFIP paper).

```
(all x -security_violation(x)).
(person(c) & in_the_building(c)).
guard(a).
(all x (security_escort(x) -> -escorts(x,c))).
(all x all y ((security_badge(x) & has(c,x)) ->
(surrender(c,y) & has(a,y))))).
```

----- PROOF -----

```
61 [] -visitor(x) | -in_the_building(x) |
security_escort($f13(x)).
62 [] -visitor(x) | -in_the_building(x) |
escorts($f13(x),x).
100 [] in_the_building(c).
102 [] -security_escort(x) | -escorts(x,c).
105 [] visitor(c).
112 [hyper,105,62,100] escorts($f13(c),c).
113 [hyper,105,61,100] security_escort($f13(c)).
147 [hyper,102,113,112] .
```

% Query 1.

```
(exists x all y (employee(x) & in_the_building(x) &
security_badge(y)
& -has(x,y))).
```

----- PROOF -----

```
1 [] -employee(x) | person(x).
39 [] -person(x) | -security_badge(y)
| -display(x,y) | has(x,y).
64 [] -person(x) | -in_the_building(x)
| display(x,$f14(x)).
98 [] employee($c6).
99 [] in_the_building($c6).
100 [] security_badge(y).
101 [] -has($c6,y).
106 [hyper,98,1] person($c6).
114 [hyper,106,64,99] display($c6,$f14($c6)).
120 [ur,101,39,106,100] -display($c6,x).
121 [binary,120,114] .
```

% Query 2. Note: The following does not cause a conflict since visitor is not in the building.

```
(exists x all y all z (visitor(x) & security_badge(y) &
display(x,y)
& in_the_building(x) & -security_violation(z))).
(exists x all y all z (visitor(x) & security_badge(y) &
display(x,y)
& -security_violation(z))).
```

----- PROOF -----

```
1 [] -employee(x) | person(x).
39 [] -person(x) | -security_badge(y)
| -display(x,y) | has(x,y).
```

```
64 [] -person(x) | -in_the_building(x)
| display(x,$f14(x)).
98 [] employee($c6).
99 [] in_the_building($c6).
100 [] security_badge(y).
101 [] -has($c6,y).
106 [hyper,98,1] person($c6).
114 [hyper,106,64,99] display($c6,$f14($c6)).
120 [ur,101,39,106,100] -display($c6,x).
121 [binary,120,114] .
```

% Query 3. Note: inside omitted; conflict due to two badges. Next statement includes inside.

```
(exists x exists y exists z all w (visitor(x) &
security_badge(y) & security_badge(z)
& has(x,y) & has(x,z) & display(x,y)
& -display(x,z) & -security_violation(w))).
(exists x exists y exists z all w (visitor(x) &
security_badge(y) & security_badge(z)
& in_the_building(x) & -security_violation(w) & has(x,y)
& has(x,z) & display(x,y)
& -display(x,z))).
```

----- PROOF -----

```
2 [] -visitor(x) | person(x).
43 [] -person(x) | -security_badge(y)
| -security_badge(z) | -has(x,y) | -has(x,z) | (y = z) |
security_violation($f7(x,y,z)).
98 [] visitor($c8).
99 [] security_badge($c7).
100 [] security_badge($c6).
101 [] has($c8,$c7).
102 [] has($c8,$c6).
103 [] display($c8,$c7).
104 [] -display($c8,$c6).
118 [] -security_violation(z).
119 [hyper,98,2] person($c8).
168 [ur,102,43,119,99,100,101,118] ($c7 = $c6).
234 [para_from,168,103] display($c8,$c6).
235 [binary,234,104] .
```

% Query 4. Note: As desired, the following does not produce a conflict. However, 187 clauses

```
% were needed before a conclusion was reached.
(employee(c) & in_the_building(c)).
(all y (security_escort(y) -> -escorts(y,c))).
```

----- PROOF -----

```
6 [] -employee(x) | -visitor(x).
98 [] employee(c).
101 [] visitor(c).
106 [ur,98,6] -visitor(c).
107 [binary,106,101] .
```

% Query 5.
(exists x exists y all z (employee(x) & security_badge(y)
& loses(x,y)
& -security_violation(z))).

----- PROOF -----
1 [] -employee(x) | person(x).
67 [] -person(x) | -security_badge(y)
| -loses(x,y) | security_violation(\$f16(x,y)).
98 [] employee(\$c7).
99 [] security_badge(\$c6).
100 [] loses(\$c7,\$c6).
105 [] -security_violation(z).
106 [hyper,98,1] person(\$c7).
114 [ur,106,67,99,105] -loses(\$c7,\$c6).
115 [binary,114,100] .

% Query 6.
(exists x all y exists w exists z exists t (visitor(x) &
guard(z) & employee(w)
& security_badge(t) & has(x,y) & at_entry(x) &
at_entry(z) & visiting(x,w) & expired_id(y)
& shows_id_to(x,y,z) & issued(x,t))).

----- PROOF -----
11 [] -invalid_id(x) | -valid_id(x).
32 [] -expired_id(x) | invalid_id(x).
55 [] -visitor(w) | -security_badge(x)
| -issued(w,x) | valid_id(\$f11(w,x)).
98 [] visitor(\$c6).
101 [] security_badge(\$f29(y)).
106 [] expired_id(y).
108 [] issued(\$c6,\$f29(y)).
115 [hyper,106,32] invalid_id(x).
123 [ur,115,11] -valid_id(x).
139 [ur,101,55,98,123]
-issued(\$c6,\$f29(x)).
140 [binary,139,108] .

% Query 7.
(exists x exists y exists z (visitor(x) &
exiting_the_building(x) & security_badge(y)
& has(x,y) & displays(x,y))).

----- PROOF -----
2 [] -visitor(x) | person(x).
45 [] -person(x) | -security_badge(y)
| -surrender(x,y) | -displays(x,y).
74 [] -visitor(w) | -exiting_the_building(w)
| -security_badge(x) | -has(w,x) | surrender(w,x).
98 [] visitor(\$c8).
99 [] exiting_the_building(\$c8).
100 [] security_badge(\$c7).
101 [] has(\$c8,\$c7).
102 [] displays(\$c8,\$c7).
107 [hyper,98,2] person(\$c8).
123 [hyper,101,74,98,99,100] surrender(\$c8,\$c7).

131 [ur,102,45,107,100]
-surrender(\$c8,\$c7).
132 [binary,131,123] .

% Query 8.
(exists x all z (employee(x) &
-first_day_occur(x) & in_the_building(x)
& -security_violation(z))).

----- PROOF -----
1 [] -employee(x) | person(x).
39 [] -person(x) | -security_badge(y)
| -display(x,y) | has(x,y).
63 [] -person(x) | -in_the_building(x) |
security_badge(\$f14(x)).
64 [] -person(x) | -in_the_building(x) |
display(x,\$f14(x)).
80 [] -employee(w) | -employee(y)
| -security_badge(x) | -has(u,x)
| -display(w,x)
| security_violation(\$f24(w,u,x)).
98 [] employee(\$c6).
100 [] in_the_building(\$c6).
105 [] -security_violation(z).
106 [hyper,98,1] person(\$c6).
112 [hyper,106,64,100] display(\$c6,\$f14(\$c6)).
113 [hyper,106,63,100] security_badge(\$f14(\$c6)).
133 [hyper,112,39,106,113] has(\$c6,\$f14(\$c6)).
135 [ur,112,80,98,98,113,105]
-has(\$c6,\$f14(\$c6)).
136 [binary,135,133] .

% Query 9.
(exists x exists y exists z exists u (person(x) &
security_badge(y) & security_dept(z) & holiday(u)
& obtain_badge_from_on(x,y,z,u))).

----- PROOF -----
12 [] -holiday(x) | day(x).
15 [] -regular_business_day(x)
| -holiday(x).
78 [] -person(x) | -security_badge(w)
| -security_dept(z) | -day(u)
| -obtain_badge_from_on(x,w,z,u)
| regular_business_day(u).
98 [] person(\$c9).
99 [] security_badge(\$c8).
100 [] security_dept(\$c7).
101 [] holiday(\$c6).
102 [] obtain_badge_from_on(\$c9,\$c8,\$c7,\$c6).
109 [hyper,101,12] day(\$c6).
110 [ur,101,15]
-regular_business_day(\$c6).
111 [ur,110,78,98,99,100,109]
-obtain_badge_from_on(\$c9,\$c8,\$c7,\$c6).
112 [binary,111,102] .

% Query 10.

(visitor(c) & in_the_building(c)).

(all y (visiting(c,y) -> visitor(y))).

(all z -security_violation(z)).

----- PROOF -----

2 [] -visitor(x) | person(x).

6 [] -employee(x) | -visitor(x).

39 [] -person(x) | -security_badge(y)

| -display(x,y) | has(x,y).

40 [] -person(x) | -security_badge(y)

| issued(x,y) | -has(x,y)

| security_violation(\$f5(x,y)).

58 [] -visitor(w) | -security_badge(x)

| -issued(w,x) | employee(\$f12(w,x)).

59 [] -visitor(w) | -security_badge(x)

| -issued(w,x) | visiting(w,\$f12(w,x)).

63 [] -person(x) | -in_the_building(x)

| security_badge(\$f14(x)).

64 [] -person(x) | -in_the_building(x)

| display(x,\$f14(x)).

98 [] visitor(c).

99 [] in_the_building(c).

100 [] -visiting(c,y) | visitor(y).

101 [] -security_violation(z).

102 [hyper,98,2] person(c).

106 [hyper,102,64,99] display(c,\$f14(c)).

107 [hyper,102,63,99] security_badge(\$f14(c)).

117 [hyper,106,39,102,107] has(c,\$f14(c)).

125 [ur,117,40,102,107,101] issued(c,\$f14(c)).

127 [hyper,125,59,98,107] visiting(c,\$f12(c,\$f14(c))).

128 [hyper,125,58,98,107]

employee(\$f12(c,\$f14(c))).

142 [ur,128,6] -visitor(\$f12(c,\$f14(c))).

148 [ur,142,100]

-visiting(c,\$f12(c,\$f14(c))).

149 [binary,148,127] .

% Query 11.

(exists x all y all z (visitor(x) & in_the_building(x) &

security_badge(y)

& -issued(x,y) & -security_violation(z))).

----- PROOF -----

2 [] -visitor(x) | person(x).

39 [] -person(x) | -security_badge(y)

| -display(x,y) | has(x,y).

40 [] -person(x) | -security_badge(y)

| issued(x,y) | -has(x,y)

| security_violation(\$f5(x,y)).

64 [] -person(x) | -in_the_building(x)

| display(x,\$f14(x)).

98 [] visitor(\$c6).

99 [] in_the_building(\$c6).

100 [] security_badge(y).

101 [] -issued(\$c6,y).

102 [] -security_violation(z).

103 [hyper,98,2] person(\$c6).

109 [hyper,103,64,99] display(\$c6,\$f14(\$c6)).

113 [ur,101,40,103,100,102] -has(\$c6,x).

119 [ur,113,39,103,100] -display(\$c6,x).

120 [binary,119,109] .

% Query 12. Note: no conflict as desired; % required
92 clauses to verify; set of support % empty as
expected.

% Testing if SM 15 implied by rule set.

(employee(a) & exiting(a)).

(security_badge(b) & has(a,b)

& -surrender(a,b)).

